PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

"RESOLUCIÓN DEL PROBLEMA DEL VENDEDOR VIAJERO UTILIZANDO PROGRAMACIÓN CON RESTRICCIONES"

JONATHAN ENRIQUE VIVANCO NAVARRO

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

DICIEMBRE, 2009

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

"RESOLUCIÓN DEL PROBLEMA DEL VENDEDOR VIAJERO UTILIZANDO PROGRAMACIÓN CON RESTRICCIONES"

JONATHAN ENRIQUE VIVANCO NAVARRO

Profesor Guía: **José Miguel Rubio León**

Carrera: **Ingeniería de Ejecución en Informática**

Diciembre, 2009

Dedicatoria

Dedicado a todos que me ayudaron a lo largo de esta etapa. Y dejo esta cita de Mi grupo favorito 'X Japan' de la canción 'Say Anything':

"Yo creí que si el tiempo pasaba todo se volvería hermoso.
Que sí la lluvia se detiene, las lágrimas limpiarían las cicatrices de mis recuerdos.
Todo comienza a vestirse de frescos colores, cada sonido comienza a entonar una cordial melodía. Los celos adornan una página épica, el deseo es abrazado en un sueño,
pero mi mente permanece en caos y ..."

Jonathan Vivanco Navarro.

Agradecimientos

A mis seres queridos que me ayudaron y dieron apoyo en lo largo que significo esta travesía y termino de esta etapa.

Resumen

Este trabajo pretende proponer una forma interesante de resolver problemas complejos utilizando Programación con Restricciones (CP), particularmente el Problema del Vendedor Viajero (TSP). Para ello, se presentan conceptos y definiciones de la Programación con Restricciones, pasando por técnicas, algoritmos y heurísticas, poniendo énfasis en la Satisfacción de Restricciones (CSP), pues se orienta a los problemas con dominios finitos y posteriormente con Problemas de Optimización con Satisfacción de Restricciones (CSOP). También se describe el TSP, presentando su definición, un breve repaso histórico de los intentos por resolverlo para muchas ciudades y mencionando algunos métodos surgidos/utilizados en el desarrollo hecho por los primeros investigadores. Además, se presenta una herramienta con la cual se enfrentó este desafío: Mozart/Oz, mostrando sus ventajas y características que nos ayudarán con el objetivo principal, el que es resolver el TSP y posteriormente se mostrarán unos problemas clásicos de CSP. Luego, se presentará un modelamiento del TSP como un CSOP y su adaptación a Oz, logrando elaborar un Prototipo de Sistema, pudiendo surgir alguna novedad digna de publicación.

Abstract

This report expects to suggest an interesting way to solve complex problems using Constraint Programming (CP), specifically the Travelling Salesman Problem (TSP). To manage this goal, here are showed concepts and definitions of Constraint Programming, as well the techniques, algorithms and heuristics, emphasizing the Constraint Satisfaction Problems (CSP), because this directs to the problems with finite domains and later with Constraint Satisfaction Optimization Problem (CSOP). Here is described the TSP too, presenting its definition, a short historic revision of the attempts to solve it oriented to a lot of cities and naming some methods emerged/used in the development done by the first researchers. Besides, here is presented a tool which is used to face this challenge: Mozart/Oz, showing its advantages and features which will help us managing the main goal, which is solve the TSP as a CSOP and its adaptation to Oz, managing to elaborate a System Prototype, maybe emerging a newness worthy of publication.

Índice

Lista de Abreviaturas o Siglas

CP:	Programación con Restricciones.
CBJ:	Conflict-Directed Backjumping
CSP:	Programación con Satisfacción de Restricciones
CSOP:	Problema de Optimización con Satisfacción de Restricciones
B&B:	Branch and Bound
BJ:	BackJumping
BT:	Backtracking Cronológico
FC:	Forward Checking
GAs:	Algoritmos Genéricos
MC:	Heurística Maximun Cardinality
MD:	Heurística Maximun Degree
MDV:	Heurística Maximun Domiain Values
MFV:	Heurística Forward Checking
MRV:	Heurística Remaining Values
MW:	Heurística Minimun Width
QAP:	Problema de Asignación Cuadrática
QTk:	Graphical User Interface Desing for Oz
TSP:	Problema del Vendedor Viajero

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Introducción

En este mundo existen infinitos problemas e infinitas posibilidades de alcanzar, en forma teórica y práctica, soluciones a dichos problemas. Sin embargo, dada la naturaleza de algunos de ellos, es posible encontrarse con un alto número de alternativas posibles para intentar encontrar su(s) solución(es). Es en este aspecto, que desde hace un buen tiempo, la informática ha estado disponible para ayudar a encontrar los caminos con mayor rapidez y, en algunos casos, encontrar la(s) respuesta(s).

Para que un sistema informático sepa qué hacer ante parámetros de entrada para entregar alguna respuesta adecuada, es necesario diseñar y programar algoritmos que permitan recorrer caminos y soluciones posibles para su evaluación, ya sea por fuerza bruta, o algún método heurístico.

Académicos, profesionales, investigadores, entre otros, han aportado su granito de arena para la aparición de algoritmos y técnicas, utilizadas tanto en informática como en otras áreas, cuyo interés principal es resolver problemas complejos en importantes campos del quehacer mundial. Es así como los problemas de asignación, problemas de encontrar rutas más cortas, problemas de optimización, entre otros, ya no tienen carácter de ser imposibles de optimizar. Es en éste ámbito, en el que nos moveremos para el presente trabajo, enfocándonos en un caso conocido: "Problema del Vendedor Viajero", explicando que se puede resolver usando el paradigma de programación con restricciones, qué implica utilizar dicho paradigma, qué herramientas de apoyo y qué algoritmos existen para ir en busca de las soluciones, y más adelante demostrar de manera práctica lo investigado.

El TSP es un clásico problema de optimización de tipo NP-Hard (NP-duro) [39]. Cuando se refiere a un problema de clase de complejidad NP, indicamos que existe un algoritmo que resuelve el problema en un tiempo polinómico pero solamente en una máquina no determinista, por lo cual la solución en una máquina normal el tiempo se hace exponencial y por lo tanto intratable [40]. Ahora si a esta clasificación la referimos como NP-Hard se indica que el problema de clase NP es posible reducirlo a un tiempo polinomial de resolución a P. Una de las formas de permitir lo anteriormente dicho es por medio de aplicación de heurísticas, cual nos entrega un valor muy cercano al óptimo en tiempos razonables de ejecución.

En este proyecto nos centraremos en el estudio de la resolución de problemas de satisfacción de restricciones (CSP), el cual se abarca con profundidad en el capítulo 3 y del Problema de Optimización con Satisfacción de Restricciones (CSOP), el cuál es una extensión del CSP, abarcado con profundidad en el capítulo 5, con el fin de resolver el problema del Vendedor Viajero (TSP), abarcado con profundidad en el capítulo 4 y elaborar un prototipo de sistema que resuelva el TSP según las exigencias del cliente Neogística S.A.

1.2 Objetivos

1.2.1 Objetivos Generales

Investigar, analizar, desarrollar y evaluar soluciones teórico-prácticas basadas en programación con restricciones, para resolver el Problema del Vendedor Viajero (TSP).

1.2.2 Objetivos Específicos

- Investigar: el paradigma de Programación con Restricciones, el Problema del Vendedor Viajero, el problema de optimización con satisfacción de Restricciones y por ultimo investigar sobre el lenguaje de programación Oz, que será el utilizado en la elaboración de pruebas y desarrollo del software final del proyecto.

- Aplicar pruebas de rendimiento de las Estrategias de Numeración en los problemas de N-Reinas y el de Coloración de Mapas, utilizando el algoritmo con mejor rendimiento para aplicarlo en instancias reales del Problema del Vendedor Viajero.

- Desarrollar y evaluar un Prototipo de Sistema que permita la resolución del Problema del Vendedor Viajero.

Capítulo 2

Programación con Restricciones

2.1 Introducción

La programación con restricciones, Constraint Programming (CP), se define como el estudio de sistemas computacionales basados en restricciones cuyo objetivo es: resolver problemas mediante la declaración de restricciones sobre el área del problema y posteriormente encontrar la(s) solución(es) que las satisfagan. Para el caso de optimizar unos criterios determinados [1], se conoce como el problema de optimización de satisfacción de restricciones, el cual se tratará en el capítulo 5 del presente informe.

La programación con restricciones surgió recientemente como un área de investigación que combina la investigación de diversas áreas como: La Inteligencia Artificial, cuyos estudios empezaron a principios de la década de los setentas, el Procesamiento Simbólico, cuyos estudios empezaron a principios de la década de los ochentas y por último la Lógica Computacional [2].

La programación con Restricciones se puede separar en 2 ramas que comparten las mismas terminologías pero sus orígenes y técnicas de resolución son diferentes [1]. La primera es la Satisfacción de Restricciones (CSP), la cual es aplicada a problemas con dominios finitos, mientras que la segunda rama, la Resolución de Restricciones, está orientada principalmente en resolver problemas con dominios infinitos o dominios más complejos.

2.2 Conceptos Básicos de CSP

A continuación se describen los conceptos básicos que son necesarios para la formulación de los problemas como un CSP y los cuales se utilizaran a lo largo de este proyecto y su aplicación en el problema del vendedor viajero (TSP, véase capitulo 4).

Un problema de satisfacción de restricciones (CSP) es una terna (X, D, C) donde:

- X es un conjunto de n variables $\{ x_1, \ldots, x_n \}$.
- D = $< D_1, \ldots, D_n >$ es un vector de dominios. La i-ésima componente D_i es el dominio que contiene todos los posibles valores que se le pueden asignar a la variable x_i .
- C es un conjunto finito de restricciones.

Los objetivos de un CSP centran en encontrar: una solución sin preferencia alguna, todas las soluciones, una óptima, o al menos una buena solución. Todo esto dado a alguna función objetivo definida en términos de algunas o todas las variables.

Instanciación o Asignación

Una instanciación de un conjunto de variables es una tupla de pares ordenados, donde cada par ordenado (x, a) asigna el valor de "a" a la variable x.

Consistencia

Un valor $a_i \in D_i$ es un valor consistente para x_i sí existe al menos una solución del CSP en el cual $x_i = a_i$. El dominio mínimo de una variable x_i es el conjunto de todos los valores consistentes para la variable, es decir, quedan excluidos aquellos valores que no forman parte de ninguna solución [1].

$$\forall x_i \in X, \forall a \in D_i, x_i = a \text{ Forma parte de una solución del CSP}$$

Una tupla $((x_1, a_1), \ldots, (x_i, a_i))$ es localmente consistente si satisface todas las restricciones formadas por las variables de la tupla. En la sección 3.3.1 se entrará en profundidad el concepto de Consistencia.

Solución

Una Solución es la asignación de valores dentro de su dominio a cada variable de forma tal que se satisfaga cada restricción, es decir, una solución es una tupla consistente que contiene todas las variables del problema. De este mismo modo, una solución parcial es una tupla consistente que contiene solo algunas variables del problema. Por lo tanto un problema es consistente, si existe al menos una solución, es decir una tupla consistente $(a_1, a_2, ..., a_i)$.

Espacio de Búsqueda

El Espacio de búsqueda es el producto cartesiano de los dominios de las variables, es decir, si se tiene una conjunto de n variables $\{x_1, x_{2,}, ..., x_n\}$ cada una con dominio $D_{x_1}, D_{x_2}, ..., D_{x_n}$ respectivamente, entonces el espacio de búsqueda es igual a $D_{x_1} \times D_{x_2} \times ... \times D_{x_n}$.

El Tamaño del espacio de Búsqueda

Corresponde a la cardinalidad del producto cartesiana que forma el Espacio de Búsqueda:

$$Card\left(D_{x_1} \times D_{x_2} \times D_{x_n}\right) = Card\left(D_{x_1}\right) Card\left(D_{x_2}\right) ... Card\left(D_{x_n}\right) \qquad (2.1)$$

Espacio de Solución

Corresponde a un conjunto del espacio de búsqueda que satisface todas las restricciones.

Equivalencia

Dos problemas de satisfacción de restricciones P_1 y P_2 son equivalentes si tienen el mismo espacio de solución.

2.2.1 Notación de CSP

En general:

- El número de variables de un CSP se denota con la letra ***n***
- La longitud del dominio de una variable x_i se denota por $d_i = |D_i|$
- El numero totales de restricciones se denota con la letra ***c***

- La Aridad máxima de una restricción se denota por la letra ***k***
- En caso de problemas disyuntivos, el número máximo de disyunciones que tiene una restricción disyuntiva se denota por la letra ***l***

Variables

Para representar las variables se utiliza las últimas letras del alfabeto en cursivas, por ejemplo *x, y, z,* los subíndices se utiliza letras seleccionada por la mitad del alfabeto o números enteros, por ejemplo x_1, x_i, x_j. Al conjunto de variables $x_i, ..., x_j$ se denota por $X_{i,...,j}$.

Dominios/Valores

El dominio de una variable x_i se denota por D_i. A los valores individuales de un dominio se representa mediante las primeras letras del alfabeto, estas pueden ir seguidas de subíndices.

Restricciones

Una restricción *k*-aria entre las variables $\{x_1, ..., x_k\}$ se denota por $C_{1..k}$, cuando los índices de una restricción no son relevante se denota simplemente por *C*. El conjunto de las variables involucradas en una restricción $C_{i..k}$ se representa por $X_{C_{i...k}}$. [1][3][4]

El resto de las notaciones que aparezcan a lo largo del proyecto se definirán en su debido momento.

2.2.2 Restricciones

Una restricción puede definirse extensionalmente mediante un conjunto de tuplas validas o no validas y también intencionalmente mediante una función aritmética. En el caso de CSPs continuos es imposible representar las restricciones extensionalmente ya que hay un número infinito de tuplas validas y no validas. [1][4]

A continuación se mostrara algunas definiciones sobre restricciones con sus respectivas explicaciones de sus propiedades básicas:

Aridad

Es el número de variables que está compuesta una restricción, por ejemplo, una restricción unaria es una restricción que consta de una sola variable, de esta misma forma

una restricción binaria consta de 2 variables al igual que una restricción ternaria consta de 3 variables. Una restricción no binaria es una restricción que tiene un número arbitrario de variables (n-aria).

Ejemplo: La restricción $y \leq 25$ es una restricción unaria sobre la variable y. La restricción $y_7 - y_3 \neq 9$ es una restricción binaria. La restricción $4x_1 + 6x_2 - 5x_3 \leq 3$ es una restricción ternaria. Por lo tanto un ejemplo de restricciones n-aria sería $4x_1 + 6x_2 - 5x_3 + 9x_4 \leq 2$.

Tupla

Una tupla p de una restricción $C_{i..k}$ es un elemento del producto cartesiano $D_i \times ... \times D_k$ (anteriormente mencionado en el capitulo 3.2.1). Una tupla *p* que satisface la restricción $C_{i..k}$ se le llama tupla permitida o valida. Una tupla *p* que no satisface la restricción $C_{i..k}$ se le llama tupla no permitida o no valida. Una tupla *p* de una restricción $C_{i..k}$ se dice que es soporte para un valor $a \in D_j$ si la variable $x_j \in X_{C_{i...k}}$, p es una tupla permitida y contiene a *a* en la correspondiente posición de x_j en la restricción.

Comprobación de la consistencia

Comprobación de la consistencia es verificar si una tupla dada es permitida o no por una restricción.

A continuación nos centraremos en el CSP dado que la naturaleza del problema del TSP es de dominio finito, como se mencionó en las secciones anteriores del presentaré trabajo, no es necesario para este proyecto entrar en profundidad en el concepto de Resolución de Restricciones.

2.3 Resolución del Problema de Satisfacción de Restricciones (CSP)

La resolución de un problema de satisfacción de restricciones (CSP) consta de dos fases diferentes:

- Modelar el problema como un problema de satisfacción de restricciones. La modelización expresa el problema mediante una sintaxis de CSPs, es decir, mediante un conjunto de variables, dominios y restricciones del CSP.

- Procesar el problema de satisfacción de restricciones resultante. Una vez formulado el problema como un CSP, hay dos maneras de procesar las restricciones:

1. Técnicas de consistencia. Se trata de técnicas para la resolución de CSPs basadas en la eliminación de valores inconsistentes de los dominios de las variables.

2. Algoritmos de búsqueda. Estos algoritmos se basan en la exploración sistemática del espacio de búsqueda hasta encontrar una solución o probar que no existe tal solución.

Las técnicas de consistencia o inferenciales permiten deducir información del problema, (niveles de consistencia, valores posibles de variables, dominios mínimos, etc.), aunque en general se combinan con las técnicas de búsqueda, ya que reducen el espacio de soluciones y los algoritmos de búsqueda exploran dicho espacio resultante. [1][4]

2.3.1 Técnicas de consistencia

Como se menciono anteriormente en el capítulo 3.3, son técnicas para la resolución de CSPs basadas en la eliminación de valores inconsistentes de los dominios de las variables.

Una de las principales dificultades que se podría encontrar en los algoritmos de búsqueda es la aparición de inconsistencias locales que van apareciendo continuamente.

Las inconsistencias locales son valores individuales o combinaciones de valores de las variables que no pueden participar en la solución porque no satisface alguna propiedad de consistencia. [1][4][5].

Las técnicas de consistencia local borran valores inconsistentes de las variables, es decir, formar nuevas restricciones implícitas que nos facilita la reducción del espacio de búsqueda. Estas técnicas son usadas como etapas de proceso donde se detectan y eliminan las inconsistencias locales antes de empezar o en el transcurso de la búsqueda con el fin de reducir el árbol de búsqueda [4].

A continuación se presentara los diversos niveles de inconsistencia locales recopilados de los siguientes referencias [4] [1] [6] [7] [8]:

- Un problema es (*i,j)-consistencia* (Freuder la presentó como la noción genérica de consistencia) si cualquier solución a un subproblema con *i* variables puede ser extendida a una solución incluyendo *j* variables adicionales. Cuando i es k - 1 y j es 1, podremos obtener la k-consistencia.

- Un CSP n-ario es (*i,j)-consistente* si y solo si $\forall x_i \in X, D_i \neq \varnothing$ y cualquier instanciación consistente de *i* variables puede ser extendido a una instanciación consistente que involucra a *j* variables adicionales.

- Un CSP binario es path-consistente si y solo si $\forall x_i, x_j \in X, (x_i, x_j)$ es path-consistente. Un par de variables (x_i, x_j) es path-consistente si y solo si $\forall (a,b) \in C_{ij}, \forall x_k \in X, \exists c \in D_k$ por lo que *c* es una soporte para *a* en c_{ik} y *c* es un soporte para b en c_{jk}.

- Un CSP binario es fuertemente path-consistente si y solo si es *(j,1)-consistente* para $j \leq 2$.

- Un CSP binario es inversamente path-consistente si y solo si $\forall (x_i, a) \in D, \forall x_j, x_k \in X$ tal que $j \neq i \neq k \neq j, \exists (x_j, b) \in D$ y $(x_k, c) \in D$ tal que *b* es soporte para *a* en C_{ij}, c es un soporte para *a* en C_{ik} y *c* es un soporte para b en C_{jk}.

- Un CSP binario es vecino inversa-consistente si y solo si $\forall (x_i, a) \in D, (x_i, a)$ se puede extender a una instanciación consistente de los vecinos de x_i.

- Un CSP binario es path-consistente restringido si y solo si $\forall x_i \in X, D_i \neq \varnothing$, D_i es arco-consistente y, $\forall (x_i, a) \in D_i, \forall x_j \in X$ tal que (x_i, a) tiene un único soporte *b* en $D_j, \forall x_k \in X$ tal que $\{C_{ik}, C_{jk}\} \in C, \exists c \in D_k$ tal que *c* es un soporte para *a* en C_{ik} y *c* es soporte para *b* en C_{jk}.

- Un CSP binario es simple arco-consistente i y solo si $\forall x_i \in X, D_i \neq \varnothing$ y $\forall (x_i, a) \in D, P\big|_{D_{x_i}=\{a\}}$ tiene un sub-dominio arco-consistente. Denotamos $P\big|_{D_{x_i}=\{a\}}$ como el CSP obtenido restringiendo D_i al valor *a* en un CSP o, donde $x_i \in X$.

- Un CSP n-ario es arco-consistente generalizado si y solo si $\forall D_i \in D, D_i \neq \varnothing$ y D_i es arco-consistente generalizado. Un dominio D_i es arco-consistente generalizado si y solo si $\forall a \in D_i, \forall x_j, ..., x_k \in X$, con $C_{j,...,i,...,k} \in C$ existe una tupla t = {b,…,a,…,c} permitida por $C_{j,...,i,...,k}$ tal que *t* es un soporte para (x_i, a) en $C_{j,...,i,...,k}$.

2.3.1.1 Consistencia de Nodo

Un problema es nodo-consistente si y solo si todas sus variables son nodo-consistentes: [1] [3] [9]

$$\forall x_i \in X, \forall C_i, \exists a \in D_i : a \text{ satisface } C_i \tag{2.2}$$

Alcanzar este nivel de consistencia nos asegura que todos los valores en el dominio de una variable satisfacen todas las restricciones unarias sobre esa variable.

Ejemplo:

$$\begin{aligned} &\text{X} = \{x_1, x_2\} \\ &D_{x_1} = \{2, 5, 8, 12\} \\ &D_{x_2} = \{1, 2, 5, 7, 9, 12\} \\ &\text{C} = \{x_1 \leq 8, x_2 \geq 2, x_2 \neq x_1\} \end{aligned}$$

Como se puede apreciar el CSP planteado en el ejemplo anterior no es nodo-consistente, este debido, que tanto nodo x_1 y el nodo x_2 no satisfacen la consistencia del nodo-consistente, debido que ambos poseen valores en su dominios que no satisfacen las restricciones unitarias $x_1 \leq 8$ y $x_2 \geq 2$ respectivamente.

Para el CSP del ejemplo expuesto anteriormente alcancé este nivel de consistencia, tendríamos que eliminar los valores del dominio de las variables x_1 y x_2 que no satisfagan las restricciones unitarias, quedando de la siguiente forma:

$$\begin{aligned} &\text{X} = \{x_1, x_2\} \\ &D_{x_1} = \{2, 5, 8\} \\ &D_{x_2} = \{2, 5, 7, 9, 12\} \\ &\text{C} = \{x_1 \leq 8, x_2 \geq 2, x_2 \neq x_1\} \end{aligned}$$

2.3.1.2 Consistencia de Arco

Dos variables (x_i, x_j) son arco-consistentes si para cada valor a en D_i hay al menos un valor *b* en D_j tal que se satisface la restricción existente entre x_i, x_j. Por lo tanto, se puede apreciar que una restricción c_{ij} implica una restricción simétrica c'_{ji}. Un CSP es arco-consistente si cada par de variables del CSP es arco-consistente:

$$\forall c_{ij} \in C, \forall a \in D_i, \exists b \in D_j : (a, b) \in c_{ij} \tag{2.3}$$

Provocando que cualquier valor en el dominio D_i de la variable x_i que no sea arco-consistente es eliminado de D_i, ya que no puede formar parte de solución alguna. Los algoritmos de arco-consistencia recorren todos los arcos de la red, por lo que tienen un coste cuadrático respecto al número de variables *n*. [1]

En la Figura 2.1 se muestra un algoritmo que asegura el nivel de arco-consistencia:

```
Procedimiento AC(X, n, D, C)  //Devuelve el CSP arco-consistemte
Inicio
Q ← C       //Inicializa los arcos del CSP
mientras Q ≠  0 hacer
     c_{i,j} ←ELIMINAR_PRIMERO(Q) // c_{i,j} restricción entre x_i, x_j
     si Revisar(c_{i,j}) entones
          para todo x_k ∈ VECINOS[ x_i ] hacer
               añadir c_{k,i} a Q
          fin para
     fin si
fin mientras
fin AC;

Función Revisar(c_{i,j}) //Retorna T si y solo si eliminamos valores de D_i
inicio
eliminar ← falso;
para todo x ∈ DOMINIO[ x_i ] hacer
     si ningún_valor y ∈ DOMINIO[ x_j ] satisface c_{i,j} entonces
          eliminar x ∈ DOMINIO[ x_i ];
          eliminar ← verdadero;
     fin si;
fin para
devolver eliminar
fin Revisar;
```

Figura 2.1: Algoritmo que asegura el nivel de arco-consistencia

La arco-consistencia dirigida es un caso particular de la arco consistencia [9], donde dado un orden linear $\rightarrow$ sobre las variables consideradas, se requiere la existencia de soportes solo en una dirección, por lo tanto, solo se requiere que unas de las siguientes condiciones sea necesario ser revisadas:

- $\forall a \in D_{x_1} \exists b \in D_{x_2} : (a,b)$ satisface C, proporcionando el orden $x_1 \rightarrow x_2$

- $\forall b \in D_{x_2} \exists a \in D_{x_1} : (a,b)$ satisface C, proporcionando el orden $x_2 \rightarrow x_1$

2.3.1.3 Consistencia de caminos

La consistencia de caminos o senda-consistencia (en inglés, Path-Consistency), es un nivel más alto de consistencia local que el arco-consistencia. La consistencia de caminos requiere que, para cada asignación de dos variables (x_i, a), (x_j, b) consistente con la restricción directa que exista entre ellas c_{ij}, exista también un valor para cada variable a lo largo del camino entre ellas de forma que todas las restricciones a lo largo del camino se satisfagan. [9]

Montanari demostró que un CSP satisface la consistencia de caminos si y solo si todos los caminos de longitud dos cumplen la consistencia de caminos. [9] En otras palabras, si y solo si cualquier solución local entre dos variables, es decir, consistente con la restricción entre ellas, es extensible a cualquier tercera variable:

$$\forall (a,b) \in c_{ij}, \forall x_k \in X, \exists c \in D_k, (a,c) \in c_{ik} \wedge (b,c) \in c_{jk} \qquad (2.4)$$

Obtiene un CSP senda-consistente para CSP binarios. Más concretamente, el algoritmo acota las restricciones del CSP inicial, de forma que el CSP resultante cumple la consistencia de caminos.

2.3.1.4 K-consistente, Consistencia Global

Un problema es fuertemente k-consistente si es i-consistente para todo $i \leq k$. Un problema fuertemente k-consistente con k variables se llama globalmente consistente. La complejidad espacial y temporal en el peor caso de forzar la k-consistencia es exponencial con k. Además, cuando k $\geq$ 2, forzar la k-consistencia cambia la estructura del grafo de restricciones añadiendo nuevas restricciones no unarias. Esto hace que la k-consistencia sea impracticable cuando k es grande. [9] [1]

2.3.2 Algoritmos de búsqueda

Los Algoritmos de búsqueda se centran en explorar el espacio de búsqueda del problema. Estos algoritmos pueden ser completos, los cuales exploran todo el espacio de búsqueda en busca de una solución, o incompletos si solamente exploran una parte del espacio de búsqueda. Los métodos que exploran todo el espacio de búsqueda garantizan encontrar una solución, si existe, o demuestran que el problema no se puede resolver. La desventaja de estos algoritmos es que son muy costosos.

Algunos ejemplos de algoritmos de búsqueda completa para CSP binarios son:

- backtracking chronologic
- backjumping
- conflict-directed backtracking
- backtracking dynamic
- forward checking
- minimal forward checking

Y algoritmos híbridos como:

- forward checking con conflict-directed backtracking
- manteniendo arco-consistencia (MAC)

Los algoritmos de Backtracking es la base fundamental de los algoritmos de búsqueda sistemática para la resolución de CSPs. Los dos métodos completos más usados son el "Generar y Testear" y el Backtracking Cronológico que se explicara a continuación [4] [3] [1].

2.3.2.1 Generar y Testear (CT)

En este método se genera las posibles tuplas de instanciación de cada una de las variables de forma sistemática y después se testean sucesivamente para ver si cumple todas las restricciones del problema. La primera combinación que satisfaga todas las restricciones, será la solución al problema. Mediante este procedimiento, el número de combinaciones generado por este método es el Producto Cartesiano de la cardinalidad del dominio de las variables, es decir:

$$\prod_{i=1,n} d_i \tag{2.5}$$

Este método es inconveniente en los problemas de cualquier tamaño grande, ya que se realizan muchas instanciaciones erróneas de valores a variables que después son rechazadas en la fase de testeo. Por ejemplo, para el caso del problema de las 4-Reinas (véase sección 7.2), se generarían $4^4 = 256$ tuplas a testear. El proceso de generación requeriría $256 * 4 = 1024$ asignaciones de variable.

2.3.2.2 Backtracking Cronológico (BT)

Este método ve el proceso de búsqueda como el incremento de la extensión de las soluciones parciales, es decir, realiza una exploración en profundidad del espacio de búsqueda, instanciando sucesivamente las variables y comprobando ante cada nueva

instanciación si las instanciaciones parciales ya realizadas son localmente consistentes. Si es así, sigue con la instanciación de una nueva variable.

El orden de instanciación es el orden en que las asignaciones de variables se realizan durante la búsqueda. Se supone que este el orden es fijo y sigue el esquema de numeración de las variables. En caso de conflicto, intenta asignar un nuevo valor a la última variable instanciada, si es posible, y en caso contrario retrocede inmediatamente a la variable asignada anterior. [10] [1] [4]

```
procedimiento Backtracking(k,V[n])
//Llamada inicial: Backtracking(1,V[n])
inicio
V[k]=Selección(d_k) //Selecciona un valor de d_k para asignar a x_k
si Comprobar(k,V[n]) entonces
     si k=n entonces
          devolver V[n] //Es una solución
     si no
          Backtracking(k+1,V[n])
     fin si
si no
     si  quedan_valores(d_x) entonces
          Backtracking(k,V[n])
     si no
          si k=1 entonces
               devolver 0 //Fallo
          si no
               Backtracking(k-1,V[n])
          fin si
     fin si
fin si
fin Backtracking
```

Figura 2.2: Algoritmo del Backtracking Cronológico

El backtracking cronológico es un algoritmo ineficiente debido que tiene una visión local del problema, es decir, solo comprueba las restricciones que están formadas por la variable actual y las pasadas, he ignora la relación entre la variable actual y las futuras. Además, este algoritmo no recuerda las acciones previas y esto podría suceder que el algoritmo repetirá la misma acción varias veces innecesariamente. Para ayudar a combatir este problema, se han desarrollado algunos algoritmos de búsqueda más robustos. Estos algoritmos se pueden dividir en: **look-backward** y **look-ahead**, que describiremos a continuación.

2.3.3 Algoritmos Look-Backward

Los algoritmos Look-Backward tratan de explotar la información del problema para comportarse más eficientemente en las situaciones sin salida. Al igual que el backtracking cronológico (ver capitulo 3.3.2.2), los algoritmos Look-Backward llevan a cabo la comprobación de la consistencia hacia atrás, es decir, entre la variable actualmente instanciada y las pasadas ya instanciadas. A continuación se mostrará algunas variantes de algoritmos Look-Backward.

2.3.3.1 Backjumping (BJ)

BJ es un algoritmo parecido al backtracking cronológico que se diferencia en su comportamiento más óptimo a la hora de encontrarse en situaciones sin salida. En vez de retroceder a la variable anteriormente instanciada, BJ salta a la variable x_j más profunda, es decir más cerca de la variable actual, que está en conflicto con la variable actual x_i donde j < i.

Se dice que una variable instanciada x_j está en conflicto con una variable x_i si la instanciación de x_j evita uno de los valores en x_i, debido a la restricción entre x_j y x_i. Cambiar la instanciación de x_j puede hacer posible encontrar una instanciación consistente de la variable actual. [11] En la Figura 2.3 se muestra un ejemplo de este algoritmo.

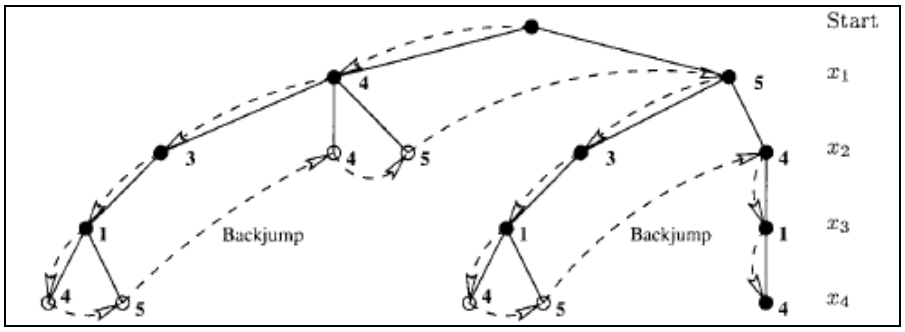

Figura 2.3: Ejemplo del Algoritmo Backjumping

2.3.3.2 Conflict-directed Backjumping (CBJ)

CBJ tiene un comportamiento de salto hacia atrás más sofisticado que BJ. Cada variable x_i tiene un conjunto conflicto formado por las variables pasadas que están en conflicto con x_i. En el momento en el que la comprobación de la consistencia entre la variable actual x_i y una variable pasada x_j falla, la variable x_j se añade al conjunto conflicto de x_i. En una situación sin salida, CBJ salta a la variable más profunda en su

conjunto conflicto, por ejemplo x_k, donde $k < i$. Al mismo tiempo se incluye el conjunto conflicto de x_i al de x_k, por lo que no se pierde ninguna información sobre conflictos. Por lo tanto CBJ necesita unas estructuras de datos más complicadas para almacenar los conjuntos conflicto. [12] En la Figura 2.4 se muestra el algoritmo de CBJ [3]:

```
Procedimiento CBJ(i)
     C_conjunto[i]←{0}
     Para_cada_un Valor en D[i]
          Asignaciones[i] ←Valor
          Consistente ← Verdad

          Para h←1 hasta i−1 Mientras Consistente
               Consistente ←Prueba(i,h)
          Si No Consistente
               C_conjunto[i]←C_conjunto[i]+{h-1}; //obtener el valor
de la última vuelta

          Si Consistente
               Si i = n
                    Mostrar – Solución()
                    C_conjunto[i]←C_conjunto[i]+{n-1}
               En otro caso
                    Retornar_profundidad ←CBJ(i+1)
                    Si Retornar_profundidad < i
                         Devolver Retornar_profundidad

     Retornar_profundidad ←Maximo(C_conjunto[i])
C_conjunto[Retornar_profundidad]←Maximo(C_conjunto[Retornar_profundid
ad], C_conjunto[i] – {Retornar_profundidad})
     Devolver Retornar_profundidad
Fin Procedimiento
```

Figura 2.4: Algoritmo de Conflict-directed Backjumping

2.3.3.3 Learning

Learning es un método que almacena las restricciones implícitas que son derivadas durante la búsqueda y las usa para podar el espacio de búsqueda. Por ejemplo, cuando se alcanza una situación sin salida en la variable x_i, entonces sabemos que la tupla de asignaciones $\left((x_1,a_1),...,(x_{i-1},a_{i-1})\right)$ nos lleva a una situación sin salida. Así podemos aprender que una combinación de asignaciones para las variables x_i, x_{i-1} no está permitida. [13]

2.3.4 Algoritmos Look-Ahead

Los algoritmos Look-Ahead hacen una comprobación inferencial hacia delante en cada instanciación, integrando un proceso inferencial durante el propio proceso de

búsqueda, por lo que también se denominan técnicas híbridas, también conocida como la propagación de los efectos de cada nueva instanciación al resto de la red.

Lo que permite lo siguiente:

- Acotar las restricciones y dominios de las variables futuras a instanciar, limitando el espacio de búsqueda pendiente.

- Encontrar las inconsistencias antes de que aparezcan, en el caso de que las instanciaciones parciales efectuadas se descubran inconsistentes con el resto de variables pendientes.

En resumen los algoritmos Look-Ahead intentan descubrir si la actual asignación localmente consistente de las k variables puede ser extendida a una solución global, provocando en caso contrario un punto de backtracking. A continuación se mostrará algunas variantes de algoritmos Look-Ahead.

2.3.4.1 Forward Checking (FC)

En cada etapa de la búsqueda, FC comprueba hacia adelante la asignación actual con todos los valores de las futuras variables que están restringidas con la variable actual, si estos valores son inconsistentes con la asignación actual son temporalmente eliminados de sus dominios. Si el dominio de una variable futura se queda vacío, la instanciación de la variable actual se deshace y se prueba con un nuevo valor. Si ningún valor es consistente, entonces se lleva a cabo el backtracking cronológico. FC garantiza que, en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura. Además cuando se asigna un valor a una variable, solamente se comprueba hacia adelante con las futuras variables con las que están involucradas, de esta forma FC puede identificar antes las situaciones sin salida y podar el espacio de búsqueda. El proceso de forward checking aplica un simple paso de arco-consistencia sobre cada restricción que involucra la variable actual con una variable futura después de cada asignación de variable. [15]

El pseudo código de algoritmo forward checking es el siguiente [1]:

1. Seleccionar x_i.
2. Instanciar $x_i \leftarrow a_i : a_i \in D_i$.
3. Razonar hacia adelante (forward-check):

    - Eliminar de los dominios de las variables $(x_{i+1},...,x_n)$ aún no instanciadas, aquellos valores inconsistentes con respecto a la instanciación (x_i, a_i), de acuerdo al conjunto de restricciones.

4. Si quedan valores posibles en los dominios de todas las variables por instanciar, entonces:

- Si i < n, incrementar i, e ir al paso (1).
- Si i = n, salir con la solución.

5. Si existe una variable por instanciar, sin valores posibles en su dominio, entonces retractar los efectos de la asignación $x_i \leftarrow a_i$. Hacer:

   - Si quedan valores por intentar en D_i, ir al paso (2).
   - Si no quedan valores:

     – Si $i > 1$, decrementar i y volver al paso (2).
     – Si $i = 1$, salir sin solución.

Forward checking con conflict-directed backjumping (FC-CBJ): Es un algoritmo híbrido que combina el movimiento hacia adelante de FC con el movimiento hacia atrás de CBJ, y de esa manera tiene las ventajas de ambos algoritmos.

2.3.4.2 Minimal forward checking (MFC)

MFC es una versión de FC que retrasa llevar a cabo toda la comprobación de la consistencia de FC hasta que sea absolutamente necesario. En vez de comprobar hacia adelante la asignación actual contra los valores de las variables futuras, MFC solo comprueba con los valores de cada variable futura hasta que se encuentra un valor que es consistente. Después, si el algoritmo retrocede llevaría a cabo las comprobaciones con los valores restantes aun no comprobados. MFC siempre lleva a cabo a lo sumo el mismo número de comprobaciones que FC. Actualmente se han demostrado que la ganancia de utilizar el MFC no supera el 10 %. [14]

2.4 Heurísticas de búsqueda

Un algoritmo de búsqueda para CSP requiere seleccionar el orden correcto del cual se van a estudiar las variables, así como el orden en el que se van a instanciar los valores de cada una de las variables permitiendo mejorar notablemente la eficiencia de la resolución, además si a esto le hacemos una ordenación adecuada de las restricciones del problema, para lograr estas mejora se utilizan heurísticas de ordenación de variables y de ordenación de valores. A continuación se mostrara las más importantes heurísticas de ordenación de variables y de ordenación de valores.

2.4.1 Heurísticas de Ordenación de Variables

Las heurísticas de ordenación de variables tratan de seleccionar lo antes posible las variables que más restringen a las demás, permitiendo poder identificar las situaciones sin salida lo antes posible y así reducir el número de vueltas atrás. La ordenación de variables puede ser estática y dinámica, las cuales se mostraran con más detalles a continuación.

2.4.1.1 Heurísticas de Ordenación de Variables Estáticas

Las heurísticas de ordenación de variables estáticas generan un orden fijo de las variables antes de iniciar la búsqueda, basado en información global derivada del grafo de restricciones inicial. A continuación se mostrara algunas heurísticas de ordenación de variables estáticas:

- **La heurística Minimum Width (MW)**

La heurística MW se basa en que la ordenación de variables corresponda a la ordenación lineal de la red que dé lugar a su menor anchura (ancho de la red). Mediante este orden, se persigue que cuando se asigne una variable, sus padres estén ya asignados, de forma que las situaciones sin salida puedan identificarse antes y se reduzca el número de vueltas atrás. [4]

- **La heurística Maximun Degree (MD)**

La heurística MD ordena las variables en un orden decreciente de su grado en el grafo de restricciones. El grado de un nodo se define como el número de nodos que son adyacentes a él. El objetivo de MD es encontrar un orden de anchura mínima, aunque no lo garantiza. [1]

- **Mínimum Domain Variable (MDV)**

La heurística MDV selecciona las variables de acuerdo a la menor cardinalidad de su dominio, es decir, las variables con dominio más pequeño son elegidas antes. [4]

Estas heurísticas anteriormente mencionadas son inferiores a la heurística del algoritmo Minimum Remaining Values (MRV) [16], que es una heurística de ordenación (véase en el capitulo 3.4.1.2).

El problema de estas heurísticas es que no toman en cuenta los cambios en los dominios o relaciones de las variables causados por la propagación de las restricciones durante la búsqueda. Debido a estos son generalmente se utilizan en algoritmos de

comprobación hacia atrás (o look-backward) donde no se lleva a cabo la propagación de las instanciaciones que se van realizando.

2.4.1.2 Heurísticas de Ordenación de Variables Dinámicas

Las heurísticas de ordenación de variables dinámicas pueden cambiar el orden de las variables dinámicamente basándose en información local que se genera durante la búsqueda cada vez que requiere la instanciación de una variable. La ordenación se basa en las instanciaciones ya realizadas y en el estado de la red en cada momento de la búsqueda.

Estas heurísticas se basan en el principio de primer fallo que sugiere que: "para tener éxito deberíamos intentar primero donde sea más probable que falle". De esta manera las situaciones sin salida pueden identificarse antes y además se ahorra espacio de búsqueda. A continuación se mostraran algunas de estas heurísticas:

- **La heurística Remaining Values (MRV)**

MRV se basa en la intuición de que si una variable tiene pocos valores de elección en su dominio, entonces es más difícil encontrar un valor consistente. MRV selecciona, en cada paso, la variable con el dominio de instanciación más pequeño.

Cuando se utiliza MRV junto con algoritmos de comprobación hacia atrás (véase capitulo 3.3.3), equivale a una heurística estática que ordena las variables de forma ascendente con la talla del dominio antes de llevar a cabo la búsqueda, y resulta igual por tanto a la heurística MDV (véase capitulo 3.4.1.1).

Cuando MRV se utiliza en conjunción con algoritmos look-ahead (véase capitulo 3.3.4), la ordenación se vuelve dinámica. En estos casos, cada nueva instanciación de variable es propagada al resto de la red y puede acotar los dominios de las variables que quedan por instanciar. Así, en cada paso de la búsqueda, la próxima variable a asignar es la variable con el dominio más pequeño. [4]

- **La heurística Maximun Cardinality (MC)**

Esta heurística selecciona la primera variable arbitrariamente y después en cada paso, selecciona la variable que es adyacente al conjunto más grande de las variables ya seleccionadas. [16].

2.4.2 Heurísticas de Ordenación de Valores

Estas heurísticas tienen como objetivo seleccionar el valor de la variable actual que más probabilidad tenga de llevarnos a una solución, la mayoría de estas heurísticas tratan de seleccionar el valor que menos reduce el número de valores útiles para las futuras variables, o alternativamente, el que deja los dominios mayores. Esto sigue la intuición de

que un sub-problema es más probable que tenga solución cuantos más valores tengan las variables que quedan por instanciar en sus dominios. A continuación se mostraran las más utilizadas:

- **La heurística Min-Conflicts**

Esta heurística ordena los valores de acuerdo a los conflictos que generan con las variables aún no instanciadas. Esta heurística asocia a cada valor *a* de la variable actual, el número total de valores en los dominios de las futuras variables adyacentes con la actual que son incompatibles con *a*. El valor seleccionado es el asociado a la suma más baja.

- **La heurística Max-Domain-Size**

Esta heurística selecciona el valor de la variable actual que deja el máximo dominio en las variables futuras.

- **La heurística Weighted-Max-Domain-Size**

Esta heurística especifica una manera de romper empates en el método anterior, en el caso de que existan varios valores de la variable actual que dejen el mismo máximo dominio en las variables futuras.

- **La heurística Point-Domain-Size**

Esta heurística asigna un peso (unidades) a cada valor de la variable actual dependiendo del número de variables futuras que se quedan con ciertas tallas de dominios.

Capítulo 3

Problema del Vendedor Viajero

3.1 Introducción y definición

El problema del vendedor viajero (en adelante TSP por sus siglas en inglés de "Traveling Salesman Problem"), se puede definir genéricamente como: "determinar la manera en como visitar o recorrer un conjunto de ubicaciones sin repetirlas, volviendo a su ubicación inicial, minimizando el camino recorrido".

Para un vendedor viajero, se trata del orden en que debe visitar las ciudades que necesita recorrer, minimizando la distancia a cubrir.

Los costos son simétricos en el sentido de que viajar desde la ciudad X a la ciudad Y tiene el mismo costo que viajar desde la ciudad Y a la ciudad X. La condición de visitar todas las ciudades implica que el problema se reduce a decidir en qué orden las ciudades van a ser visitadas.

El enunciado del problema casi decepciona, de no ser que el hecho de intentar encontrar las soluciones para un número de ciudades sobre 20, ya se torna algo imposible por métodos tradicionales. Basta con intentar enumerar los caminos posibles entre n ciudades, expresando matemáticamente su número como $(n-1)!/2$, y sobre ese encontrar la ruta optima, lo que ya nos da una idea de la dificultad a la que nos enfrentamos.

3.2 Historia

El siglo XIX comienza a suscitar interés de este problema particular, pero es en el siglo XX cuando el mundo científico pone sus ojos en él. Precisamente en el año 1954 cuando George Dantzig, Delbert Ray Fulkerson y Selmer M. Johnson [19], hicieron su aporte expresando el problema como Programación Linear Entera [20] y desarrollaron el método de Corte de Planos (cutting-plane method) [21] para encontrar la solución. De esta manera encontraron una solución óptima para una instancia de 49 ciudades (escogieron las capitales de los estados de EEUU en esa fecha -48- mas Washington D.C: – dato curioso: Alaska y Hawai se convirtieron en estados en el año 1959). Más tarde en el año 1962, Michael Held y Richard M. Karp [22], introducen heurísticas basadas en programación dinámica, las que más adelante serían utilizadas.

En la década de los 70, varios científicos lograron avances importantes utilizando "Corte de Planos" y "Poda y ramificación" (branch and bound [23]), encontrando solución

para instancias de hasta 2329 ciudades. En los años 90, Applegate, Bixby, Chvátal, y Cook, desarrollaron el software Concorde [24] con el cual se han obtenido los "records" de soluciones con grandes cantidades de ciudades (1994 – solución para 7397, 2005 – solución para 33810 ciudades, 2006 – solución para 85900 ciudades), basándose en el trabajo de Dantzig, Fulkerson y Johnson y su método:"Corte de planos".

Año	Autores	Número ciudades
1954	Dantzig, Fulkerson y Johnson	49
1971	Held y Karp	64
1975	Camerini, Fratta y Maffioli	67
1977	Grötschel	120
1980	Crowder y Padberg	318
1987	Padberg y Rinaldi	532
1987	Grötschel y Holland	666
1987	Padberg y Rinaldi	2,392
1994	Applegate, Bixby, Chvátal y Cook	7,397
1998	Applegate, Bixby, Chvátal y Cook	13,509
2001	Applegate, Bixby, Chvátal y Cook	15,112
2004	Applegate, Bixby, Chvátal, Cook, y Helsgaun	24,978
2005	Cook, Espinoza y Goycoolea	33,810
2006	Dantzig, Fulkerson y Johnson	85,900

Tabla 3.1 Record del TSP en el tiempo

En la práctica se puede utilizar el TSP para:

- Recolección de Teléfonos Públicos.
- Ruta de Camión del repartidor.
- Circuitos Integrados.
- Tiempo de setup de Máquinas.
- Secuenciamento de genes.
- Ordenamiento de observaciones en telescopios (NASA).
- Diseño de chips.
- Tour Mundial.
- Vehicle Routing.
    - Bus Escolar.
    - Atención de Llamadas de Emergencia.
    - Servicio de Correo Expreso.

¿Podemos enumerar las soluciones y escoger la mejor?

- 10 ciudades aproximadamente $10^{5,5}$ posibilidades.
- 100 ciudades aproximadamente 10^{156} 10156 posibilidades.
- 1,000 ciudades aproximadamente 10^{2565} posibilidades.
- 33,810 ciudades aproximadamente $10^{10138411}$ posibilidades.

Considerando que la Edad del universo es aproximadamente 10^{16} segundos, que el Número de átomos en el universo es aproximadamente menor a 10^{100}. Por lo tanto la Enumeración solo es posible para problemas número de ciudades muy pequeños.

La heurística de Held-Karp tiene una garantía de $n^2 2^n$ para el caso general. Si asumimos que las distancias son euclidianas hay heurísticas con garantía de $\frac{1}{2}$, cuando las distancias son euclidianas buenas soluciones heurísticas están dentro del 1-5% del óptimo.

3.3 Resolviendo TSP

Para resolver el TSP, se puede utilizar diversos modelados, como por ejemplo grafos o tablas de distancias (matriz de costos), con el objetivo de representar un inicio, un fin y el costo a minimizar.

Dado que los trabajos anteriores que han sido exitosos, han utilizado programación lineal, expresaremos el problema en la misma notación, que es compatible con CSP, El modelo quedaría [38] [41]:

Un ciclo, es aquel camino que satisface las condiciones, es decir, pasa por todas las ciudades al menos una vez y solo una vez, y comenzando y terminando en la misma ciudad. También tenemos que un ciclo interno (o sub-ciclo), comienza y termina en la misma ciudad, pero no pasa por todas las ciudades.

Supongamos que se debe visitar las ciudades 1,2,3…N para todo $i \neq j$, se tiene C_{ij} = la distancia que hay al ir desde ciudad i a ciudad j y sea C_{ii} un número muy alto para impedir que el algoritmo considere que puede pasar por la misma ciudad nuevamente. Además, definamos X_{ij} = { 0, 1 } en donde 1 indica que la solución pasa por ir desde i a j y 0 en caso contrario.

Función Objetivo a Minimizar es:

$$Min \quad \sum_{i=1} \sum_{j=1} c_{ij} x_{ij} \tag{3.1}$$

Las restricciones:

Para garantizar que se llega a cada ciudad exactamente una vez:

$$\sum_{i=1}^{i=N} x_{ij} = \quad 1 \qquad (j = 1...N) \tag{3.2}$$

Para garantizar que se sale de cada ciudad exactamente una vez:

$$\sum_{j=1}^{j=N} x_{ij} = \quad 1 \qquad (i = 1...N) \tag{3.3}$$

Sin embargo estas restricciones no bastan para garantizar que se está optimizando sobre recorridos, es decir que las soluciones factibles son sólo recorridos. Esto es porque permiten la existencia de sub-recorridos: Por ejemplo, en el caso de seis ciudades haciendo 1 las variables $x_{0,1}, x_{1,2}, x_{2,6}, x_{3,4}, x_{4,5}$ y $x_{5,3}$ se satisfacen todas estas restricciones. Esta solución no es un recorrido sino dos sub-recorridos: $0 \to 1 \to 2 \to 6 \to 0$ y $3 \to 4 \to 5 \to 3$.

Para restringirnos a recorridos hay que añadir restricciones adicionales. Hay varias formas de hacer esto.

Una forma:

En cualquier recorrido, para cada subconjunto de índices de N = {0, 1, . . . , n} debe haber un arco que vaya a su complemento y otro que venga. En general, para cualquier $L \subseteq N$ con $2 \le |L| \le n-1$ las restricciones

$$\sum_{\{(i,j) \mid i \in L, j \in N \setminus L\}} x_{i,j} \ge 1 \tag{3.4}$$

son satisfechas por todo tour pero todo sub-ciclo viola al menos una de ellas.

Otra forma: (complementaria)

Para cualquier $L \subseteq N$ con $2 \le |L| \le n-1$ poner las restricciones:

$$\sum_{\{(i,j) \mid i \in L, j \in L\}} x_{i,j} \ge |L| - 1 \tag{3.5}$$

Para cualquiera de estas dos formas, se necesita alrededor de 2^{n+1} restricciones, una por cada elección de L, para evitar los sub-ciclos, que resulta un número bastante grande para *n* razonable.

Entonces una estrategia muy común es ignorar estas restricciones que evitan sub-recorridos y resolver la relajación resultante. Si se obtiene un recorrido la solución es óptima, si no entonces la solución tiene un sub-recorrido: se añade una restricción que evite dicho sub-recorrido y se resuelve el nuevo problema. Se sigue así hasta encontrar una solución óptima. Esta estrategia ha sido aplicada de diversas formas, siendo efectiva para la resolución de este problema. Siendo la estrategia relajación-restricción en el espíritu de planos cortantes. [41]

Una tercera y distinta formulación del TSP: (Tucker)

Esta formulación modela las restricciones para evitar sub-recorridos como sigue (las primeras 2(n + 1) restricciones quedan igual). Se añaden variables continuas $u_i \left(i = 2...N\right)$ la cual indica la posición de la ciudad i en el ciclo y las restricciones:

$$\begin{aligned} u_i - u_j + Nx_{ij} &\leq N-1 \quad \left(i \neq j; i = 2...N;\ j = 2...N\right) \\ x_{ij} &= \{0,1\} \quad \left(i = 1...N;\ j = 1...N\right) \\ u_j &\geq 0 \qquad \left(j = 1...N\right) \end{aligned} \tag{3.6}$$

Ellas aseguran que:

- Cualquier conjunto de valores de X_{ij} que formen ciclos internos, no sean soluciones factibles.
- Cualquier conjunto de valores de X_{ij} que formen un ciclo, sea solución factible.

Para ilustrar el funcionamiento del modelo, supongamos que se tiene como solución al problema del ejemplo $x_{1,5} = x_{2,1} = x_{3,4} = x_{4,3} = x_{5,2} = 1$. Esta asignación contiene 2 ciclos internos: 1 – 5 – 2 – 1 y 3 – 4 – 3. Si escogemos el ciclo interno que no contiene a la Ciudad 1 (3 – 4 – 3) y escribimos las restricciones (3.6) correspondiente a estas asignaciones se obtiene:

$$u_3 - u_4 + 5x_{3,4} \leq 4 \quad \text{y} \quad u_4 - u_3 + 5x_{4,3} \leq 4$$

Sumando ambas restricciones se obtiene:

$$5\left(x_{3,4} + x_{4,3}\right) \leq 8$$

La restricción anterior no se puede satisfacer para la combinación $x_{4,3} = x_{3,4} = 1$, por lo tanto el sub ciclo 3 - 4 - 3 no es una solución factible para el modelo de programación lineal. Se puede verificar que las restricciones (3.6) son violadas por cualquier sub ciclo posible.

Supongamos ahora que la Ciudad 1 fue la primera visitada. Consideremos: $t_i =$ posición en el ciclo cuando la ciudad i es visitada. Luego, si $u_i = t_i$ cualquier ciclo que satisface las restricciones (3.6). A modo de ejemplo, consideremos el ciclo 1 - 3 - 4 - 5 - 2 - 1. Entonces, las otras variables son: $u_1 = 1$, $u_2 = 5$, $u_3 = 2$, $u_4 = 3$ y $u_5 = 4$. Consideremos

ahora cualquier restricción (3.6) que contenga un $x_{i,j}$ = 1. Por ejemplo, la restricción correspondiente a $x_{5,2}$ es:

$$u_5 - u_2 + 5x_{5,2} \leq 4$$

Como la Ciudad 2 es la que sigue inmediatamente a la Ciudad 5 se tiene:

$$u_5 - u_2 = -1$$

Por lo tanto, la restricción (3.6) de $x_{5,2}$ se satisface ya que:

$$-1 + 5 \leq 4$$

Consideremos ahora una restricción correspondiente a un $x_{i,j}$ = 0, por ejemplo $x_{3,2}$ de acuerdo al ciclo escogido. La restricción de $x_{3,2}$ queda:

$$u_3 - u_2 + 5x_{3,2} \leq 4 \quad \rightarrow \quad u_3 - u_2 \leq 4$$

Como las variables u_i representan la posición en el ciclo, se tiene que $u_i \leq 5$ y $u_i > 1$ $(i = 2,...N)$. En este caso:

$$\left.\begin{array}{l} u_3 \leq 5 \\ u_2 > 1 \end{array}\right\} \rightarrow \; u_3 - u_2 \leq 5 - 2 = 3$$

Por lo tanto, las restricciones (3.6) para las variables $x_{i,j}$ = 0 también se satisfacen cuando las variables contienen un ciclo.

Luego, el modelo efectivamente elimina cualquier secuencia de N ciudades que comiencen en la Ciudad 1 y que contengan un sub ciclo. Por lo tanto, el modelo resuelve el problema del Vendedor Viajero. Evidentemente, el modelo crece rápidamente según aumente el número de ciudades incluidas en el problema, por lo tanto tiende a ser poco eficiente.

En la práctica las dos primeras formulaciones han tenido más éxito que la última, debido que se tiende a pensar que si se tiene más restricciones, es más fácil obtener el óptimo. Pero las primeras dos formas están contenida en la de la última (Tucker).

3.4 Algoritmos para TSP

Las restricciones que fastidian por su cantidad y estructura son las de destrucción de sub-recorridos. Lo que queda sin ellas es un simple Problema de Asignación (flujo en redes) que se resuelve fácilmente. Entonces es natural que los algoritmos de Branch And

Bound (B&B) o de planos cortantes para el TSP utilicen algún esquema de relajación-restricción que siga estas líneas. Los algoritmos modificados relajan estas restricciones y resuelven la relajación (asignación). Si la solución obtenida es un recorrido. Caso contrario, añadir restricción que destruya algún sub-recorrido (si se hace directo es planos cortantes, si se hace a través de cotas y en sub-grafos es B&B) y proceder con la nueva relajación. Estos algoritmos son computacionalmente aplicables sólo a problemas con unos pocos cientos de ciudades.

- **Cutting-Plane Method**

Dantzig, Fulkerson, y Johnson, cuando desarrollaron el método de corte de planos, su objetivo no era encontrar la solución completa, ya que no podían, más bien se dedicaron a hacer lo que si podían: resolver usando simplex [24]. Una de las particularidades del método simplex, es que la solución óptima (si es que graficamos) siempre se va a encontrar en un extremo del conjunto de soluciones posibles.

En otras palabras, la idea es –mediante iteraciones- introducir restricciones que permitan acotar el conjunto solución en busca de un nuevo óptimo, realizando cortes al plano generado por la gráfica del conjunto de soluciones posibles.

- **Branch and Bound**

Ramificación y poda, es un algoritmo que se basa en dibujar como "árbol" las soluciones parciales en busca de soluciones en paralelo. Es decir, al dividir o ramificar posibles caminos que lleven a una solución (branch), se van introduciendo cotas que permiten descartar o podar (bound) soluciones que se alejen de los mejores resultados obtenidos por otras ramas. Este algoritmo es utilizado en las aulas de las universidades en asignaturas como Investigación Operativa, para resolver problemas de asignación y de transporte, principalmente porque es posible diseñar problemas que se puedan resolver en pocas iteraciones. Problemas con menos de 20 nodos pueden ser resueltos en forma eficiente por este método. En el Capitulo 4.3 Resolviendo CSOP usando B entraremos en más detalles sobre este algoritmo.

3.5 Heurísticas para el Problema del Vendedor Viajero

Para problemas de TSP más grandes se necesitan métodos heurísticos. Estas heurísticas se pueden clasificar en dos tipos: Las que construyen recorridos y las que los mejoran.

3.5.1 Las heurísticas que construyen recorridos

El seudo código es el siguiente:

1. Encontrar un sub-recorrido inicial (basta que tenga dos ciudades pero puede ser de más).

2. Seleccionar una ciudad a ser añadida al sub-recorrido, usando alguna regla de selección.
3. Insertar el nodo correspondiente en el sub-recorrido, basándose en un criterio de inserción.

4. Si se tiene un recorrido, parar. Caso contrario ir a 2.

Las distintas reglas de selección y criterios de inserción dan lugar a diferentes heurísticas. Veamos una de ellas: La heurística de inserción más barata.

Se comienza con un sub-recorrido T con dos ciudades i_1 e i_2, tales que:

$$d_{i_1,i_2} + d_{i_2,1} = \min_{i \neq j} \left(d_{i,j} + d_{j,i} \right) \tag{3.7}$$

es decir, el sub-recorrido más corto entre dos ciudades. Luego se busca el costo mínimo de inserción en el sub-recorrido:

$$c_{k^*} = \min_{(i,j) \in T} \left(d_{i,k} + d_{k,j} - d_{i,j} \right) \tag{3.8}$$

y se selecciona un nodo k^* con ese costo. Luego se inserta el nodo entre las ciudades involucradas en el cálculo de c_{k^*}. Finalmente se repite este proceso (con los sub-recorridos que se van generando) hasta construir un recorrido.

Ejemplo de la heurística de inserción más barata

Consideremos el TSP dado por la siguiente matriz de distancias:

$$D = \begin{bmatrix} - & 2 & 11 & 10 & 8 & 7 & 6 & 5 \\ 6 & - & 1 & 8 & 8 & 4 & 6 & 7 \\ 5 & 12 & - & 11 & 8 & 12 & 3 & 11 \\ 11 & 9 & 10 & - & 1 & 9 & 8 & 10 \\ 11 & 11 & 9 & 4 & - & 2 & 10 & 9 \\ 12 & 8 & 5 & 2 & 11 & - & 11 & 9 \\ 10 & 11 & 12 & 10 & 9 & 12 & - & 3 \\ 7 & 10 & 10 & 10 & 6 & 3 & 1 & - \end{bmatrix}$$

En primer lugar tenemos que encontrar el sub-recorrido más barato con dos ciudades. Esto se hace calculando $\min_{i \neq j} \left(d_{i,j} + d_{j,i} \right)$ que corresponde a sumar elementos simétricos de la matriz D y ver cuál nos da menor. En este caso el más pequeño es $d_{7,8} + d_{8,7} = 3 + 1 = 4.$

Calculemos ahora el mínimo costo de inserción: $c_{k^*} = \underset{(i,j)\in T}{mín}\left(d_{i,k} + d_{k,j} - d_{i,j}\right)$

k	$d_{7,k} + d_{k,8} - d_{7,8}$	$d_{8,k} + d_{k,7} - d_{8,7}$
1	$10+5-3=12$	$7+6-1=12$
2	$11+7-3=15$	$10+6-1=15$
3	$12+11-3=20$	$10+3-1=12$
4	$10+10-3=17$	$10+8-1=17$
5	$9+9-3=15$	$6+10-1=15$
6	$12+9-3=18$	$3+11-1=13$

donde el mínimo valor es 12 y se obtiene de tres formas. Seleccionamos la inserción del nodo 1 entre los nodos 8 y 7. Las restantes iteraciones se muestran en Tabla 3.2:

Iteración	Sub-recorrido	k c_k i_k j_k	Iteración elegida
2	$1\to 7\to 8\to 1$	2 2 1 7 3 8 1 7 3 8 8 1 4 12 1 7 5 10 8 1 6 8 8 1	k c_k i_k j_k 2 2 1 7
3	$1\to 2\to 7\to 8\to 1$	3 –2 2 7 4 10 2 7 5 10 8 1 6 8 8 1	k c_k i_k j_k 3 –2 2 7
4	$1\to 2\to 3\to 7\to 8\to 1$	4 14 8 1 5 10 8 1 6 8 8 1 6 8 2 3	k c_k i_k j_k 6 8 8 1
5	$1\to 2\to 3\to 7\to 8\to 6\to 1$	4 1 6 1 5 5 8 6	k c_k i_k j_k 4 1 6 1
6	$1\to 2\to 3\to 7\to 8\to 6\to 4\to 1$	5 1 4 1	k c_k i_k j_k 5 1 4 1

Tabla 3.2: Ejemplo de las interacciones generadas por la heurística Inserción más Barata

Dando como recorrido final: $1\to 2\to 3\to 7\to 8\to 6\to 4\to 5\to 1$. Este recorrido, casualmente, es óptimo. No siempre se corre con la misma suerte. Note que en la tercera columna se calcula, en cada fila y para cada k, el mínimo costo de inserción de ese k

particular (en algunos casos hay empate y el k se repite), y luego se toma el mínimo de todos ellos.

3.5.2 Las heurísticas que mejoran recorridos

Se comienza con un recorrido que se supone no óptima, por ejemplo el resultado de una heurística como la explicada arriba. Se intenta luego mejorar sucesivamente ese recorrido mediante el intercambio de aristas. Se borran $k\,(k \geq 2)$ aristas del recorrido actual y se añaden k nuevas aristas para formar un nuevo recorrido. Nos concentramos en TSP simétrico ya que la descripción es más limpia. El algoritmo más simple es el siguiente (que intercambia 2 aristas):

1. Encontrar una solución inicial para el PAV (recorrido).

2. Evaluar todos los intercambios de dos aristas de la forma que muestra la Figura 3.1

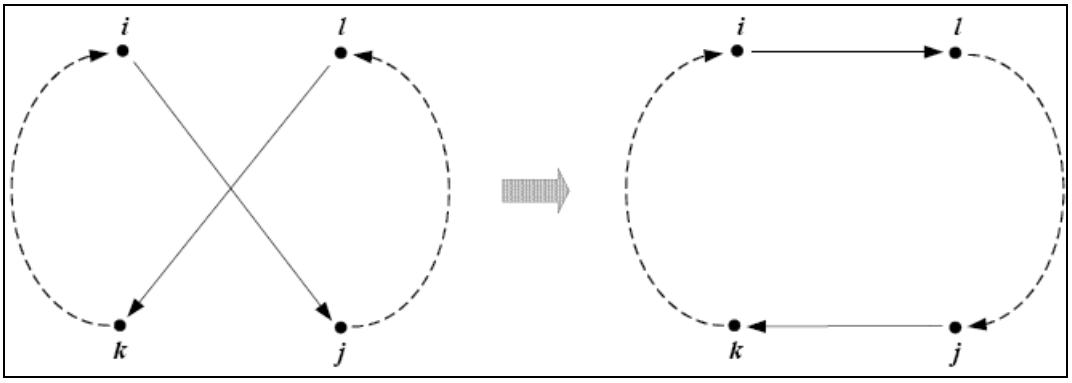

Figura 3.1 :Intercambio de Aristas heurísticas que mejoran recorridos

donde la línea punteada representa un camino. Note que el camino entre j y l invierte su sentido, lo cual no es problema porque el PAV se supone simétrico.

3. Si el intercambio que produce la mayor mejora en realidad mejora, hacer el cambio e ir a 2. En caso contrario parar.

Ejemplo: Consideremos el TSP simétrico dado por la siguiente matriz de distancias:

$$D = \begin{bmatrix} - & 5 & 7 & 8 & 11 & 10 \\ & - & 6 & 3 & 5 & 9 \\ & & - & 4 & 11 & 8 \\ & & & - & 7 & 3 \\ & & & & - & 2 \\ & & & & & - \end{bmatrix}$$

y comencemos con el recorrido $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$ que tiene longitud total 34. En la Tabla 3.3 muestra las ganancias de todos los intercambios posibles de dos aristas:

i j l k	$\left(d_{i,j} + d_{l,k}\right) - \left(d_{i,l} + d_{k,j}\right)$	Ganancia
1 2 3 4	$(5+4)-(7+3)$	-1
4 5	$(5+7)-(8+5)$	-1
5 6	$(5+2)-(11+9)$	-13
2 3 4 5	$(6+7)-(3+11)$	-1
5 6	$(6+2)-(5+8)$	-5
6 1	$(6+10)-(9+7)$	0
3 4 5 6	$(4+2)-(11+3)$	-8
6 1	$(4+10)-(8+8)$	-2
4 5 6 1	$(7+10)-(3+11)$	3

Tabla 3.3: Ejemplo de las interacciones generadas por las heurísticas que mejoran el recorrido

en donde claramente el único que realmente produce ganancia es el que intercambia las aristas (4, 5) y (6, 1) por las aristas (4, 6) y (5, 1). De manera que nos queda el nuevo recorrido $1 \to 2 \to 3 \to 4 \to 6 \to 5 \to 1$ con una distancia de 31. Así necesitándose dos iteraciones más para que el algoritmo heurístico se detenga.

Capítulo 4

Problemas de Optimización con Satisfacción de Restricciones (CSOP)

4.1 Introducción y Definición

En el capítulo 2, hemos examinado las diversas técnicas para resolver CSP. Debido que el problema del vendedor viajero no es un problema del tipo CSP, esto es causado por la naturaleza del TSP, que es un problema de optimización, por lo tanto para resolver el TSP usaremos CSOP, que explicaremos a lo largo de este capítulo.

Un problema de optimización con satisfacción de restricciones, Constraint Satisfaction Optimization Problem, consiste en un CSP estándar junto con una función objetivo f a ser optimizada. El valor de la función objetivo a menudo es representado por una variable z, junto con la restricción maximizar z o minimizar z para la maximización o minimización del problema respectivamente. Es decir:

$$CSOP = \langle X, D, C, f \rangle = \langle CSP, f \rangle$$

Donde $f : S \rightarrow valor\ numérico$

Tomamos una dupla de solución T , llamaremos $f(T)$ como $f - valor$ de T . [25]

En donde su objetivo:

Encontrar la(s) mejor(es) solución(es) con respecto a f satisfaciendo C . [26]

Con el fin de encontrar la solución óptima, en primer lugar se tiene que encontrar todas las soluciones y luego comparar sus $f - valor$. Una parte de la búsqueda de espacio sólo pueden ser podadas si se puede demostrar que la mejor solución no está en él o bien no hay solución existe en el mismo (lo que implica el conocimiento acerca de las soluciones) o que él $f - valor$ en cualquier solución en el espacio de búsqueda poda es sub-óptima (lo que implica el conocimiento de la $f - valor$).

4.2 Técnicas para hacer frente al CSOP

La búsqueda de soluciones óptimas, básicamente, implica la comparación de todas las soluciones en un CSOP. Por lo tanto, las técnicas para encontrar todas las soluciones que utilizamos en CSP nos sirven para obtener una solución en CSOP, estas técnicas fueron descritas en el capítulo 3.

Técnicas para ordenar los valores son irrelevantes cuando normalmente todas las soluciones son necesarias, porque en tal caso, todos los valores deben ser examinados. Ordenar los valores será de utilidad en reunir información durante la búsqueda de estrategias, sino más bien los conjuntos se pueden descubierto en la búsqueda de un árbol en lugar de otro, y una heurística que tiene el fin de las ramas (los valores), a fin de maximizar el aprendizaje. [25]

En las siguientes secciones, vamos a introducir dos importantes métodos para hacer frente a CSOPs. Se trata de el algoritmo Branch and Bound (B&B) y los algoritmos genéticos (GAs). El primero usa heurísticas para podar el espacio fuera de búsqueda, y el segundo es un enfoque estocástico que ha demostrado ser eficaz en problemas de combinatoria.

4.3 Resolviendo CSOP usando B&B

En la solución de CSOPs, se puede utilizar una heurística de la función *f* para orientar la búsqueda. Branch and Bound (B&B), el cual en general es un algoritmo de búsqueda para encontrar soluciones óptimas haciendo uso del conocimiento sobre la función f. Se utilizan los mismos términos introducidos en CSP, pero hay que tener en cuenta que una tupla de solución no se refiere necesariamente a la solución óptima en un CSOP. B&B es una conocida técnica de ambas operaciones en la investigación y AI. Se basa en la disponibilidad de una buena heurística para estimar los mejores valores (de acuerdo con la optimización de la función) de todas las hojas en el marco de la actual rama del árbol de búsqueda. Aunque B&B no reduce la complejidad de un algoritmo de búsqueda, podría ser más eficiente que la chronological backtracking search. Sin embargo no son fiables las heurísticas necesariamente disponibles. [25]

4.3.1 Un genérico algoritmo B&B para CSOP

Para aplicar el B&B en CSOP, se necesita una función heurística h, la cual convierte cada componente de los mapas llamada CL a un valor numérico ($h : CL \rightarrow numero$), a este valor lo llamaremos como *h-valor*. Para la función h sea aceptada, el *h-valor* de cualquier CL debe ser una sobreestimado (o subestimado) de una solución de cualquier *f-valor* de cada CL en un problema de maximización o minimización.

Una variable global, que nos referiremos como el *blound*, se inicializa en menos infinito en un problema de maximización. El algoritmo de búsqueda de soluciones en una

primera forma de profundidad. Se comporta como Chronological Backtracking (ver capitulo 3.3.2.2), salvo que antes del incremento de la extensión de las soluciones parciales, se amplía para incluir una nueva variable, el *h-valor* de la variable actual se calcula. Si este *h-valor* es inferior al *blound* en un problema de maximización entonces el subárbol bajo el actual variable es podado. Siempre que una tupla de solución se encuentra, su *f-valor* se calcula. Este *f-valor* será el nuevo *blound* si y sólo si es mayor que la existente *blound* en un problema de maximización. Cuando este *f-valor* es igual o mayor que el *blound*, el recién encontrado tupla de solución se registra como uno de la, o las mejor tuplas de solución hasta ahora. Después de todas las partes del espacio de búsqueda se han buscado o podado, la mejor tuplas de solución registrados hasta ahora son soluciones de los CSOP. [25]

```
Procedimiento Branch_and_Bound(X,D,C,f,h);
/*(X,D,C) es un CSP; f es la función en la tupla de solución, el f-
valor es maximizado; h es una heurística de estimación de la
upper-bound de f-valor de la asignación de la variable*/

Inicio */Bound es una variable global que almacena el mejor f-valor
encontrado hasta el momento, Mejor_hasta_ahora es también un variable
global que almacena la mejor solución hasta el momento*/

Bound = menos_infinito;
Mejor_hasta_ahora = nulo;
BNB(X,{},D,C,f,h);
Devolver Mejor_hasta_ahora

Fin */del Procedimiento Branch_and_Bound*/

Procedimiento BNB(sin_asignar, asignación_compuesta,D,C,f,h);

Inicio

Si sin_asignar = {} hacer
  Si f(asignación_compuesta) > Bound hacer /* sólo una solución se
devuelve*/
     Bound = f(asignación_compuesta);
     Mejor_hasta_ahora = asignación_compuesta;
  Fin si
Si no
  Si (h(asignación_compuesta) > Bound) hacer
     se elige cualquier variable x de sin_asignar
Repetir
     se elige cualquier variable x de D_x;
        Borrar v de D_x;
Si (asignación_compuesta + {<x,v>} no viola ninguna restricción)
hacer
     BNB(sin_asignar - {x}, asignación_compuesta + {<x,v>},D,C,f,h);
Hasta(Dx = {});
Fin si
Fin BNB
```

Figura 4.1 : Seudo código del algoritmo Branch and Bound

Este algoritmo es complejo si se encuentra fuera del espacio de búsqueda. La eficiencia de B&B está determinado por dos factores: la calidad de la heurística función y si un buen Bound se encuentra en una etapa temprana. En un problema de maximización, si los h-valores son siempre sobre-estimaciones de los f-valores, la más cercana a la estimación es el f-valor, es decir, cuanto más pequeño es el h-valor es (sin ser menor que el f-valor), más posibilidades habrá de que una mayor parte del espacio de búsqueda sea podada. Una rama es podada por B&B, si el h-valor del nodo actual es menor que el Bound (en un problema de maximización). Esto significa que incluso con la función fija heurístico, B&B podar fuera de proporción diferente del espacio de búsqueda, si las ramas están ordenados de otra manera, debidos a los diferentes límites se puede encontrar en diferentes ramas. [25]

4.3.2 Ejemplo de Solución de CSOP utilizando B&B

Variable	Dominios	Restricciones
x_1	4, 5	
x_2	3, 5	$x_2 \neq x_1$
x_3	3, 5	$x_3 = x_2$
x_4	2, 3	$x_4 < x_3$
x_5	1, 3	$x_5 \neq x_4$

Figura 4.2: Ejemplo de un CSOP

La Figura 4.2 muestra un ejemplo de un CSOP. Las cinco variables x_1, x_2, x_3, x_4 y x_5 tienen dominios numéricos. El f-valor de la asignación es una combinación de la suma de todos los valores de las variables. La tarea es encontrar la tupla de solución con el máximo f-valor.

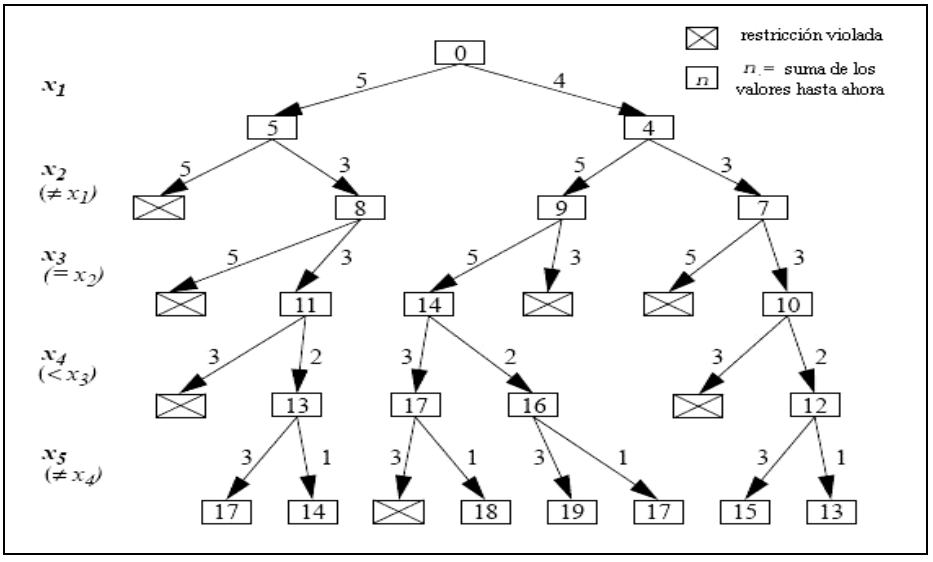

Figura 4.3: El espacio de búsqueda por simple backtracking en la solución de CSOP.

La Figura 4.3 muestra el espacio explorado por simple backtracking. Cada nodo en la Figura 4.3 representa una variable asignada, y cada rama representa la asignación de un valor a una variable no asignada. Las variables se asumen a través de la búsqueda bajo el orden x_1, x_2, x_3, x_4 y x_5. Como se explico en la sección anterior, el B&B obtendría mejor resultado si un mayor bound se encuentra antes. A fin de ilustrar el efecto de B&B, suponemos que las ramas que representan la mayor asignación de valores se buscan en primer lugar.

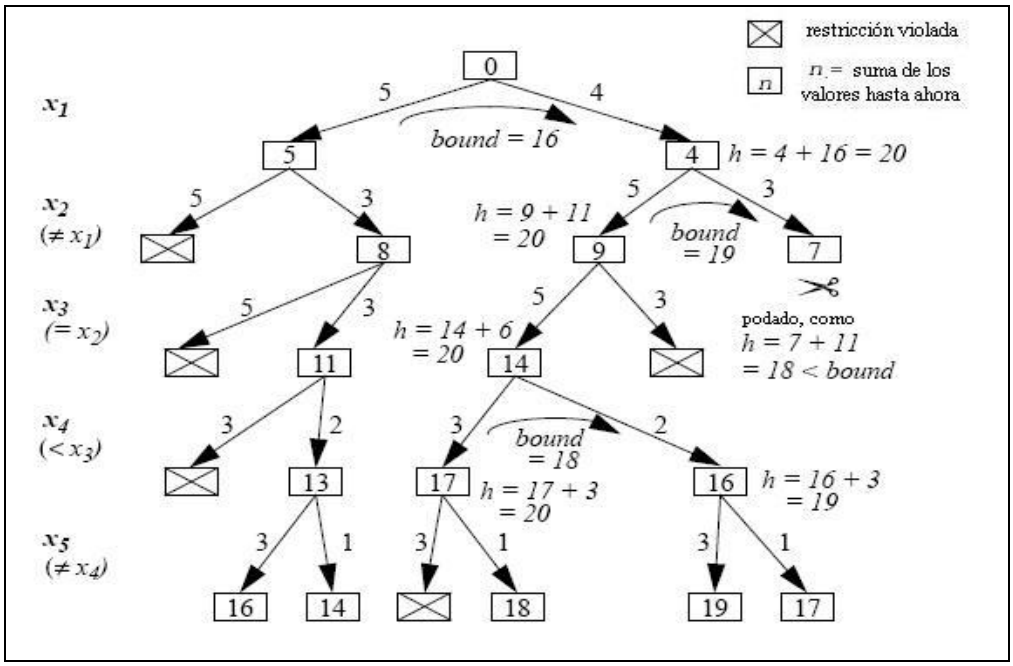

Figura 4.4: El espacio de búsqueda por Branch&Bound en la solución del CSOP.

La Figura 4.4 muestra el espacio de búsqueda por B&B bajo el mismo orden de búsqueda como un simple backtracking. El h-valor para un nodo se calcula como el valor asignado hasta la fecha, más la suma de los valores máximos para las variables de la asignación. Por ejemplo, el h-valor de $(\langle x_1,4\rangle,\langle x_2,5\rangle)$ es 4 + 5 (los valores asignados hasta la ahora) más 5 + 3 + 3 (los valores máximos que pueden asignarse a x_3, x_4 y x_5), que es 20.

Según el procedimiento Branch and Bound descrito en la sección anterior, el bound se inicializa en menos infinito. Cuando el nodo para $(\langle x_1,5\rangle\langle x_2,3\rangle\langle x_3,3\rangle\langle x_4,2\rangle\langle x_5,3\rangle)$ es alcanzado, el bound se actualiza a (5 + 3 + 3 + 2 + 3 =) 16. Este bound no tiene ningún efecto sobre la mitad izquierda de la búsqueda del árbol en este ejemplo. Cuando el nodo para $(\langle x_1,4\rangle\langle x_2,5\rangle\langle x_3,5\rangle\langle x_4,3\rangle\langle x_5,1\rangle)$ es alcanzado, el bound se actualiza a 18. Cuando el nodo para $(\langle x_1,4\rangle\langle x_2,5\rangle\langle x_3,5\rangle\langle x_4,2\rangle\langle x_5,3\rangle)$ es alcanzado, el bound se actualiza a 19. Cuando el nodo $(\langle x_1,4\rangle\langle x_2,3\rangle)$ es examinado, su h-valor (que es 18) es inferior a la actual (que es 19). Por lo tanto el subárbol del nodo $(\langle x_1,4\rangle\langle x_2,3\rangle)$ es podado. Después de esta

nodación, se llegó a la conclusión de que $\left(\langle x_1,4\rangle\langle x_2,5\rangle\langle x_3,5\rangle\langle x_4,2\rangle\langle x_5,3\rangle\right)$ es la solución óptima. En la Figura 4.4, 21 nodos que se han explorado, en contraposición a 27 nodos en la Figura 4.3.

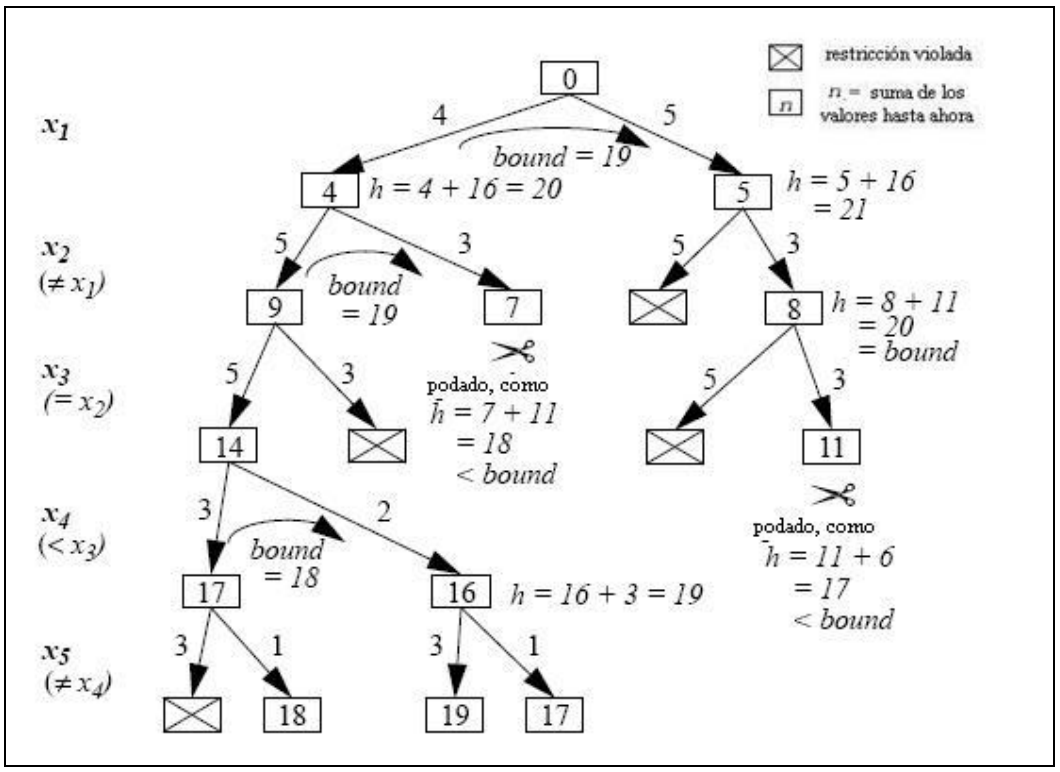

Figura 4.5: El espacio de búsqueda por Branch&Bound en la solución del CSOP.

En la Figura 4.5 se muestra la importancia de encontrar una mayor bound en una fase temprana. En la Figura 4.5, suponemos que es buscado $\langle x_1,4\rangle$ antes de $\langle x_1,5\rangle$, los demás parámetros se mantienen iguales. La solución óptima se encuentra después de 10 nodos que se han explorado. El bound 19 se utiliza para podar el subárbol abajo $\left(\langle x_1,4\rangle\langle x_2,3\rangle\right)$ (cuyo valor es 18) y $\left(\langle x_1,5\rangle\langle x_2,3\rangle\langle x_3,3\rangle\right)$ (cuyo valor es 18). Hay que tener en cuenta que el nodo para $\left(\langle x_1,5\rangle\langle x_2,3\rangle\right)$ habría sido podado si solo se hubiera requerido una sola solución, porque el valor de $\left(\langle x_1,5\rangle\langle x_2,3\rangle\right)$ es tan igual al bound. Sólo el nodo 17 ha sido explorado en la Figura 4.5.

4.4 Greedy Method

Los algoritmos que se obtienen aplicando este esquema se denominan, por extensión, algoritmos voraces. El esquema forma parte de una familia de algoritmos mucho más amplia denominada "Algoritmos de Búsqueda Local" de la que también forman parte, por ejemplo, el método del gradiente, los algoritmos Hill-Climbing, los algoritmos genéticos o los Simulated Annealing.

A continuación se caracterizan de forma general las condiciones que deben cumplir los problemas que son candidatos para ser resueltos usando un algoritmo voraces [36]:

- El problema a resolver ha de ser de optimización (Minimización o Maximización) y debe existir una función objetivo a minimizar o maximizar. En la Figura 4.6 se muestra una función objetivo típico, que es lineal y multivariable.

$$f : \mathbb{N} \times \mathbb{N} \times ... \times \mathbb{N} \rightarrow \mathbb{N}\left(\circ \Re^{+}\right)$$
$$f\left(x_1, x_2, ..., x_n\right) = c_1 \cdot x_1 + c_2 \cdot x_2 + ... + c_n \cdot x_n$$

Figura 4.6: Función Objetivo Típica para Greedy Method

- Existe un conjunto de valores posibles para cada una de las variables de la función objetivo, es decir su dominio.

- Puede existir un conjunto de restricciones que imponen condiciones a los valores del dominio que pueden tomar las variables de la función objetivo.

- La solución al problema debe ser expresable en forma de secuencia de decisiones y debe existir una función que permita determinar cuándo una secuencia de decisiones es solución para el problema, es decir la función solución. Entendemos por decisión la asociación a una variable de un valor de su domino.

- Debe existir una función que permita determinar si una secuencia de decisiones viola o no las restricciones, la función factible.

Esta es una caracterización muy genérica y pero también sirve para los problemas que pueden ser resueltos usando Backtracking o Branch&Bound.

El propósito de un algoritmo voraz es encontrar una asociación de valores a todas las variables tal que el valor de la función objetivo sea óptimo. Para ello sigue un proceso secuencial en el que a cada paso toma una decisión (decide qué valor del dominio le ha de asignar a la variable actual) aplicando siempre el mismo criterio (función de selección). La decisión es localmente óptima, es decir, ningún otro valor de los disponibles para esa variable lograría que la función objetivo tuviera un valor mejor, y luego comprueba que la

nueva decisión junto con todas las tomadas anteriormente no violan las restricciones y así consigue una nueva secuencia de decisiones factible. [36]

En el siguiente paso el algoritmo voraz se encuentra con un problema idéntico, pero estrictamente menor, al que tenía en el paso anterior y vuelve a aplicar la misma función de selección para tomar la siguiente decisión. Esta es, por tanto, una técnica descendente.

Pero nunca se vuelve a reconsiderar ninguna de las decisiones tomadas. Una vez que a una variable se le ha asignado un valor localmente óptimo y que hace que la secuencia de decisiones sea factible, nunca más se va a intentar asignar un nuevo valor a esa misma variable.

El coste de este algoritmo depende de dos factores [36]:

1. del número de iteraciones, que a su vez depende del tamaño de la solución construida y del tamaño del conjunto de candidatos.

2. del coste de las funciones selección y factible, ya que el resto de las operaciones del interior del bucle pueden tener coste constante. El coste de la función factible dependerá del número y complejidad de las restricciones del problema. La función de selección ha de explorar el conjunto de valores candidatos y obtener el mejor en ese momento y por eso su coste depende, entre otras cosas, del tamaño del conjunto de candidatos. Para reducir el coste de la función de selección, siempre que sea posible se preprocesa el conjunto de candidatos antes de entrar en el bucle. Normalmente el preproceso consiste en ordenar el conjunto de valores posibles lo que hace que el coste del algoritmo voraz acabe siendo del orden de $\theta(n * \log n)$, con n el tamaño del conjunto a ordenar.

El hecho de utilizar un algoritmo voraz para obtener la solución de un problema no garantiza que la solución obtenida sea la óptima pero, por el contrario e independientemente de que la consigan o no, el coste invertido es pequeño ya que el algoritmo sólo genera una de entre todas las posibles secuencias de decisiones. Y como las decisiones localmente óptimas no garantizan que siempre se vaya a obtener la combinación de valores que optimiza el valor de la función objetivo, el óptimo global, siempre habrá que intentar demostrar que la solución obtenida es la óptima. La inducción es la técnica de demostración más utilizada para estos algoritmos.

4.5 Algoritmos Genéticos

Como CSPs, CSOPs son NP-duros por la naturaleza. A menos que un algoritmo de B&B dispone de una heurística que da estimaciones bastante precisas de los f-valores, es poco probable que sea capaz de resolver problemas muy grandes. Además, las buenas heurísticas de funciones no siempre están disponibles, especialmente cuando la función a optimizar no es una simple función lineal.

Algoritmos genéticos (GAs) son una clase de algoritmos de búsqueda estocástica que basan sus ideas de la evolución en la naturaleza. Se han desmostado a lo largo de la literatura que GAs es eficaces en una serie de bien conocidas y ampliamente los problemas de optimización combinatoria, incluido el problema del vendedor viajero, el problema de asignación cuadrática (QAP), y aplicaciones como la programación. De este mismo modo se ha descubierto que el GAs podría ser útil para las grandes, pero poco limitado CSOPs donde sus soluciones sin llegar a ser óptimas son aceptables.

La idea de GAs se basa en la evolución, cuando mas individual es una persona, más posibilidades tienen de sobrevivir y reproducirse y transmitir sus genes a las generaciones futuras. A la larga, los genes que contribuyen positivamente a la aptitud de un individuo tendrán una mejor oportunidad de permanecer en la población.

Cuándo usar estos algoritmos

Los algoritmos genéticos son de probada eficacia en caso de querer calcular funciones no derivables o de derivación muy compleja, aunque su uso es posible con cualquier función.

Deben tenerse en cuenta también las siguientes consideraciones:

- Si la función a optimizar tiene muchos máximos/mínimos locales se requerirán más iteraciones del algoritmo para "asegurar" el máximo/mínimo global.

- Si la función a optimizar contiene varios puntos muy cercanos en valor al óptimo, solamente podemos "asegurar" que encontraremos uno de ellos (no necesariamente el óptimo).

4.5.1 Funcionamiento de un algoritmo genético básico

Para aplicar esta idea a la optimización de problemas, primero hay que ser capaz de representar a los candidatos de solución como una cadena de bloques de construcción. En algunas aplicaciones de otros GAs, un candidato solución está representado por un conjunto de cadenas en lugar de una solo string. Cada elemento debe tener un valor de un dominio finito. Muchas investigaciones se centran sobre el uso de bloques de construcción binario, es decir, bloques de construcción que sólo puede tomar en 0 o 1 como sus valores. Para aplicar a problemas de optimización de GAs, uno también debe ser capaz de expresar la función de optimización en el problema en función de los valores adoptadas por la construcción de bloques en una cadena.

La optimización de la función, que no es necesariamente lineal, en el GAs se refiere como la función de evaluación. Un string es análogo a un cromosoma en las células biológicas, y los bloques de construcción son análogos a los genes. Los valores por los

bloques se llaman *allels*. El valor de la cadena devuelta por la función de evaluación se refiere a la aptitud de la cadena.

Para aplicar GAs a problemas de optimización, una población de candidatos de soluciones se genera y se mantiene. Es decir, los string pueden ser generados de forma aleatoria en el proceso de inicialización. A más sofisticados de inicialización puede garantizar que todos los allels de todos los bloques de construcción están presentes en la población inicial. El tamaño de la población es uno de los parámetros de GAs que han de ser fijado por el programa de diseño. Después de la población inicial se genera, la población se le permite evolucionar dinámicamente. Un uso común de control de flujo es el *canónico* GAs, que se muestra en la Figura 4.7. [25]

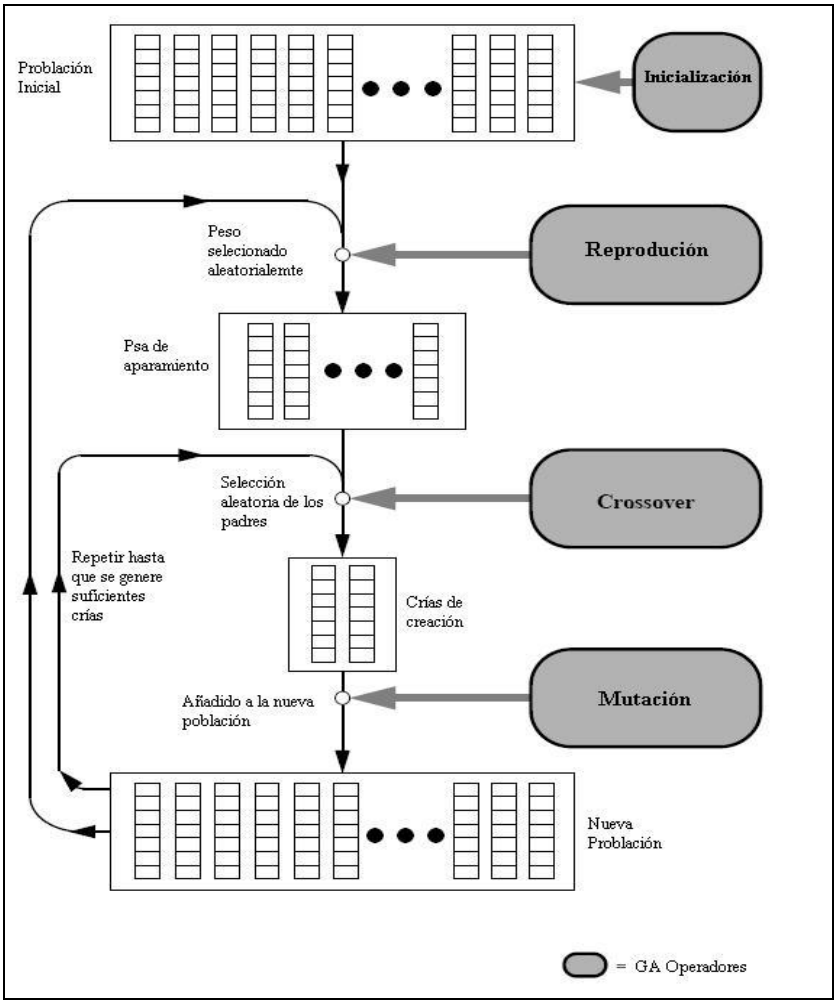

Figura 4.7: Control de flujo y de las operaciones en el Algoritmo Genérico Canónico

Un algoritmo genético puede presentar diversas variaciones, dependiendo de cómo se aplican los operadores genéticos (cruzamiento, mutación), de cómo se realiza la selección y de cómo se decide el reemplazo de los individuos para formar la nueva población. En general, el pseudocódigo consiste de los siguientes pasos: [25]

- **Inicialización:** Se genera aleatoriamente la población inicial, que está constituida por un conjunto de cromosomas los cuales representan las posibles soluciones del problema. En caso de no hacerlo aleatoriamente, es importante garantizar que dentro de la población inicial, se tenga la diversidad estructural de estas soluciones para tener una representación de la mayor parte de la población posible o al menos evitar la convergencia prematura.

- **Evaluación:** A cada uno de los cromosomas de esta población se aplicará la función de aptitud para saber cómo de "buena" es la solución que se está codificando.

- **Condición de término** El GAs se deberá detener cuando se alcance la solución óptima, pero ésta generalmente se desconoce, por lo que se deben utilizar otros criterios de detención. Normalmente se usan dos criterios: correr el GAs un número máximo de iteraciones (generaciones) o detenerlo cuando no haya cambios en la población. Mientras no se cumpla la condición de término se hace lo siguiente:

    - **Selección** Después de saber la aptitud de cada cromosoma se procede a elegir los cromosomas que serán cruzados en la siguiente generación. Los cromosomas con mejor aptitud tienen mayor probabilidad de ser seleccionados.

    - **Cruzamiento** El cruzamiento es el principal operador genético, representa la reproducción sexual, opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.

    - **Mutación** modifica al azar parte del cromosoma de los individuos, y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.

    - **Reemplazo** una vez aplicados los operadores genéticos, se seleccionan los mejores individuos para conformar la población de la generación siguiente.

4.5.2 Aplicando Algoritmo Genéticos en CSOP

Como la mayoría de los algoritmos estocásticos de búsqueda, el GAs no garantiza la obtención de soluciones óptimas. Sin embargo, muchos problemas de la vida real son difíciles de completar con métodos. Para este tipo de problemas, casi óptimas soluciones a menudo son aceptables si se pueden generar en el tiempo disponible. GAs ofrecen esperanza en la solución de esos problemas. Un GA son unas formas para resolver CSOPs porque el GAs han tenido éxito en muchos problemas de optimización y las tuplas de solución en CSPs naturalmente puede estar representado por las cadenas de GAs, como se explica a continuación.

Para un CSOP con n variables, cada cadena se puede utilizar para representar una n-tupla compuestas, donde los bloques de construcción representan las variables (en orden fijo), cada uno de los cuales pueden tener un valor de un dominio finito. Cada esquema de GA representa un compuesto en una variable CSP.

Por ejemplo, si hay cinco variables en el CSOP, x_1, x_2, x_3, x_4, x_5, a continuación, una cadena de cinco bloques de construcción se utilizará para representar a GAs en el compuesto 5-variables en el CSOP. Si las variables se dan por encima de los pedidos en una representación GAs, entonces el compuesto variable $(\langle x_1, a\rangle\langle x_3, b\rangle)$ estaría representado por el esquema $a*b**$, donde * representa un comodín.

Quienes no puedan ser representados explícitamente en una cadena son las restricciones en un CSOP. Para garantizar que el GAs generado garantice que su compuesto de variable se ajuste a las restricciones, se puede usar unas de las siguientes estrategias:

- Asegúrese de que la población contiene sólo las cadenas que cumplan las restricciones, a través del diseño de la inicialización, los operadores de cruce y las mutaciones apropiadas.

- Construir dentro de la función de evaluación una función castigadora que asigna valores a la baja aptitud de cadenas que infrinjan las restricciones. Esto reduce la posibilidad de que las cadenas que violan ciertas restricciones se reproduzcan.

4.6 Técnica elegida a utilizada para el desarrollo del Prototipo del Sistema

A lo largo de este capítulo se describió el CSOP y sus diversas aplicaciones y se mostro las diversas técnicas complementarias que se pueden utilizar para resolver el TSP, dado esto en este proyecto utilizaremos la técnica B&B, debido a su facilidad de integración y adaptación con respecto al modelamiento del TSP como un problema de CSOP en el lenguaje de programación Mozart/Oz, del cual entraremos en más detalles en el siguiente capítulo.

Hay que destacar que mediante la integración de B&B, podemos obtener un numero de óptimos aceptables para analizarlo en el presente trabajo, siendo posible que en un trabajo a futuro se pueda ampliar el rendimiento del B&B con las heurísticas y técnicas explicadas en el presente trabajo, así obteniendo un conjunto de óptimos mas cercanos al óptimos global del TSP.

Capítulo 5

Lenguaje de Programación Mozart/Oz

5.1 Introducción

Oz es un lenguaje multi-paradigma diseñado para aplicaciones avanzadas, concurrentes, distribuidas y de tiempo real. Oz fue originalmente desarrollado en el laboratorio de Programación de Sistemas en la Universidad de Saarland por Gert Smolka y sus estudiantes a comienzos de 1990. En 1996 el desarrollo de Oz continuó en cooperación con el grupo de investigación de Seif Haridi en el Instituto Sueco de Ciencias de la Computación.

Desde 1999, Oz ha sido continuamente desarrollado por un grupo internacional, el Consorcio Mozart, que estuvo compuesto originalmente por la Universidad de Saarland, el Instituto Sueco de Ciencias de la Computación, y la Universidad Católica de Louvain. En 2005, la responsabilidad de gestionar el desarrollo de Mozart fue transferida a un grupo base, el Tablero Mozart, con el propósito expreso de abrir el desarrollo de Mozart a una comunidad mayor. El sistema de programación Mozart implementa Oz 3, el último en la familia de lenguaje multi-paradigma basados en el modelo de concurrencia restringida. La principal mejora respecto a Oz 2 son los *functors* (una especie de componente de software) y *futures* (para mejorar el comportamiento del flujo de datos). [18]

Oz es un lenguaje de programación multi-paradigma (puede usarse en forma procedural, funcional, con restricciones lógicas u orientado a objetos) que soporta programación en soft-real time, concurrencia, distribución y programación reactiva. Mozart es el sistema de computación que soporta a Oz y le permite la comunicación con el resto del mundo (se encarga de los sockets, I/O, interfaz gráfica, etc). Mozart ha sido portado a diferentes plataformas como Unix, FreeBSD, Linux, Microsoft Windows y Mac OS X. [17] [18] [35] [34]

Oz está definido en función de un lenguaje Kernel y el resto de las construcciones de éste lenguaje se pueden considerar como complemento para este Kernel. [35]

Las principales ventajas de Oz radican en la programación con restricciones y la programación distribuida, permitiendo la creación dinámica de cualquier número de hilos secuenciales, es decir, hilos de fluido de datos en el sentido que un hilo que ejecuta una operación se suspenderá hasta que todos los operandos necesarios tenga un valor bien definido. [17] [18]

5.2 Características

- Oz combina las características dominantes de la programación orientada a objetos suministrando estados, tipos abstractos de datos, clases, objetos y herencia.

- Oz provee las características sobresalientes de la programación funcional suministrando sintaxis composicional, procedimientos de primera clase y léxico de alcance. De hecho, la entidad Oz es clase primera, incluye procesos, hilos, clases, métodos y objetos.

- Oz provee las características más destacadas de la programación lógica y programación restringida suministrando variables lógicas, constructos disyuntivos y estrategias de búsqueda programable.

- Oz es un lenguaje concurrente donde los usuarios pueden crear dinámicamente cualquier número de hilos secuenciales que pueden interactuar unos con otros. Sin embargo, en contraste con los lenguajes concurrentes convencionales, cada hilo Oz es un hilo con flujo de datos. Ejecutar una expresión en Oz se sigue sólo cuando todas las dependencias reales de flujos de datos involucradas son resueltas.

- El sistema Mozart soporta distribución transparente en red de cómputos Oz, es decir, múltiples sitios de Oz pueden conectarse y automáticamente se comportan como una sola computación Oz, variables compartidas, objetos, clases y procedimientos. Los sitios se desconectan automáticamente cuando las referencias entre entidades en sitios diferentes cesan de existir.

- En un ambiente distribuido Oz se provee seguridad en el lenguaje, por lo tanto, todas las entidades del lenguaje son creadas y pasadas explícitamente. Una aplicación no puede olvidar referencias ni acceder referencias que no le han sido dadas explícitamente. La representación subyacente del lenguaje de entidades es inaccesible al programador. Esta es una consecuencia de tener un almacenamiento abstracto y alcance léxico. Junto con los procedimientos de primera clase, estos conceptos son esenciales para implementar una política basada en la capacidad de seguridad, la cual es importante en cómputos distribuidos abiertos.

5.3 El lenguaje Kernel Oz

El modelo de ejecución Oz consiste en hilos de flujos de datos observando un almacenamiento parcial. Los hilos contienen secuencias *Si* y se comunican a través de referencias compartidas en el almacenamiento. Un hilo es un flujo de datos si ejecuta su próxima expresión cuando todos los valores que la expresión necesita están disponibles. Si la expresión necesita un valor que ya no está disponible, entonces el hilo automáticamente se bloquea hasta que pueda acceder el valor. Como veremos la disponibilidad de datos en el modelo Oz es implementada usando variables lógicas. El almacenamiento compartido no es memoria física, más bien es un almacenamiento abstracto el cual sólo permite operaciones que son legales para las entidades involucradas, es decir, no hay una forma directa para inspeccionar las representaciones internas de entidades.

El almacenamiento contiene variables lógicas limitadas y sin límites, células (llamadas apuntadores mutables, es decir, estados explícitos) y procedimientos (llamados léxicamente clausuras de alcance que son las entidades de primera clase). Las variables pueden referenciar los nombres de procedimientos y células. Las células apuntan a variables. Los procedimientos de referencia externa son variables. Cuando una variable es limitada, esta desaparece, esto es, todos los hilos que la referencian automáticamente referenciarán el cambio de atadura. Las variables pueden ser limitadas a una entidad, incluyendo otras variables. Las variables y procedimientos almacenados son monotónicos, es decir, la información puede sólo ser adicionada a ellos, no cambiada ni removida. [34][35]

```
<Declaración> ::= <Declaración1><Declaración2>
X = f(I1:Y1…In:Yn)
X = <numerico>
X = <atomo>
X = <boleano>
{NewName X}
X = Y
local X1…Xn in S1 end
proc{X Y1…Yn} S1 end
{X Y1…Yn}
{NewCell X Y}
Y =@ X
X := Y
{Exchange X Y Z}
if B then S1 else S2 end
thread S1 end
try S1 catch X then S2 end
raise X end
```

Figura 5.1: El lenguaje Kernel de OZ

La Figura 5.1 define la sintaxis abstracta de la expresión S en el lenguaje kernel. La totalidad del lenguaje Oz es definido transformando todas sus expresiones en este lenguaje kernel. Oz soporta conceptos como objetos, clases, claves de reingreso y puertos. A continuación definiremos brevemente cada posible expresión.

5.4 Conceptos Básicos y Definiciones

Las **variables** de OZ son lógicas. Una vez que se ligan (bind) a un valor, éste valor no puede cambiarse. El ámbito de las variables es explícito de su declaración con *local*. En forma interactiva pueden introducirse con *declare*. [34]

Asignación de variables: es una función de mapeo de variables a números enteros. [17]

Los **procedimientos** se definen usando *proc* y se referencian luego por una variable. [34]

Las **celdas** se crean con *NewCell*, se leen con @ y se actualizan con := o con *Exchange*.

Los **condicionales** usan la construcción *if* y las **excepciones** *raise* y *try*.

Dominio Finito: es un conjunto de números enteros no negativos. La notación *m#n* se utiliza para el dominio finito *m,…,n.*

Restricción: es una fórmula de lógica de predicado.

Ejemplo: $x_1 + x_2 = 7x_3$

Problema de Dominio Finito: es un conjunto finito *P* de restricciones cuantificadas libres tales que *P* contiene una restricción de dominio para cada variable que ocurre en una restricción *P*.

Restricción de Dominio: es una restricción de la forma $x_i \in D_{x_i}$, donde D_{x_i} es un dominio finito. Estas restricciones pueden expresar restricciones de la forma $x_1 = n$ que es equivalente a $x_i \in m\#n$.

Ejemplo:

Restricciones cuantificadoras libres: $x_1 \neq x_2, x_1 \neq x_2, x_2 \neq x_3$

Restricciones de dominio: $D_{x_1} = \{5,6,7\}, D_{x_2} = \{5,6,7\}, D_{x_3} = \{5,6,7\}$

Restricción Básica: estas restricciones tienen algunas de las siguientes formas:

$x_i = n$ ó $x_i \in D_{x_i}$, donde D_{x_i} es un dominio finito.

Solución de un problema de dominio finito: es una asignación variable que satisface cada restricción en *P*. [17]

5.5 Emacs, el Ambiente de Programación Mozart

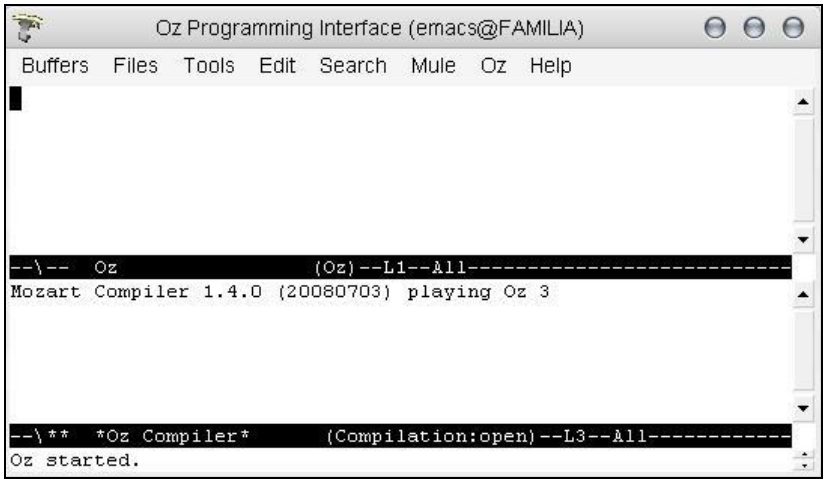

Figura 5.2: Interfaz Emacs

El ambiente de programación Mozart se presenta como una ventana *Emacs*. Cada división o ventana nueva se conoce como un *buffer* (notación *emacs*). Emacs como editor tiene un help que se invoca con ^*h t* (*Control-h* y luego la tecla *t*, en formato *emacs c-h t*) En *emcas*, las combinaciones útiles son *c-g* (*quit*, cancela el comando en proceso) y *c-x u* (*undo*), se sale con *c-x c-c*. [35] [18] [34]

Los comandos *emacs* de *oz* comienzan en general con *c-. .* Se escribe el código en el buffer llamado Oz.

```
{Browse 'Hol Mundo'}
{browse "Hola Mundo"}
{Browse 48*3}
```

Figura 5.3: Ejemplo de comando Emacs de Oz.

En la Figura 5.3 se muestra una llamada al procedimiento Browse con un dato: 48*3 (La notación clásica de cálculo lambda). Se alimenta con esta línea al compilador (c-. c-l) y apareciendo que el cuando fue aceptado en el buffer "Oz compiler".

```
local Maximo A B C in
     %primer ejemplo en Oz
     proc {Maximo X Y Z}
          if X>Y then Z=X else Z=Y end
     end
          A=4
          B=5
          {Maximo A B C}
          {Browse C}
end
```

Figura 5.4: Un ejemplo más completo de Oz

Las variables y los nombres de procedimiento deben comenzar con mayúsculas y declararse. Los comentarios comienzan con %.

Usando el lenguaje Kernel se logran las demás características de Oz como:

- Abreviaturas: if B1 then S1 elseif B2 then S2 else S3 end
- Módulos: como {For 1 11 3 Browse} que dan como resultado 1 4 7 10, Los módulos se conocen además como packages.

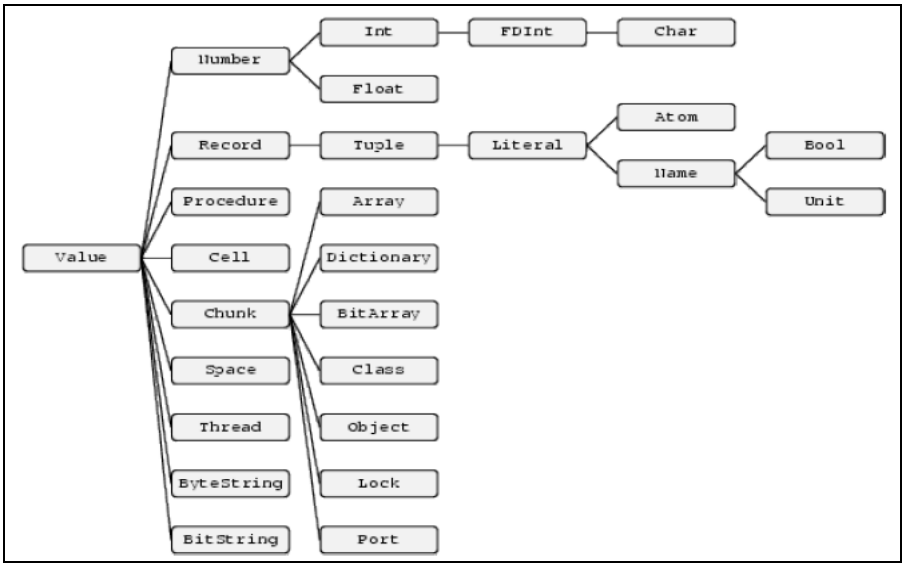

Figura 5.5: Estructura de los tipos primitivos de Oz

Los caracteres usan la codificación ISO 8859-1 (no Unicode) y su representación externa es numérica. Se introducen con & (por ejemplo, &t=116). Los records son datos estructurados.

```
local L L1 L2 v S in
     L=lista(val:5 sig:nil)
     L1=lista(V S)
     V=4
     S=nil
     L2=lista(3 L1)
     {Browse L2}
     {Browse L.val}
end
```

Figura 5.6: Ejemplo de records

En la Figura 5.6, L se define como una lista con 2 features (campos): val y sig. L1 se define como una lista en función de variables que aún no tienen valor. El segundo *Browse* muestra cómo seleccionar un campo. El procedimiento *Arity* indica la lista de las *features* tiene un record.

Las lists son las listas que implementa Oz en forma nativa. Hay dos formas de escribir una lista: Una forma *cerrada* como L = [a 2 3 4] y la otra forma es de forma *abierta* según la definición que una lista es *nil* o un *valor* seguido del símbolo | seguido de una *lista*.

```
local L1 L2 in
L1= 1|2|a|b
     {Browse L1}
     L2= {1 2 a b}
     {Browse L2}
end
```

Figura 5.7: Ejemplo de una lista abierta

La seudovariable "_ " (subrayado) denota una variable que luego no será usada.

```
local L1 L2 Largo in
     fun {Largo Ls}
          case Ls
          of nil then 0
          [] _|Lr then 1+{Largo Lr}
          end
     end
     L1= 1|2|3|4|55|66|nil
     L2= [1 2 3 4]
     {Browse {Largo L1} }
     {Browse {Largo L2} }
end
```

Figura 5.8: La función para obtener el largo

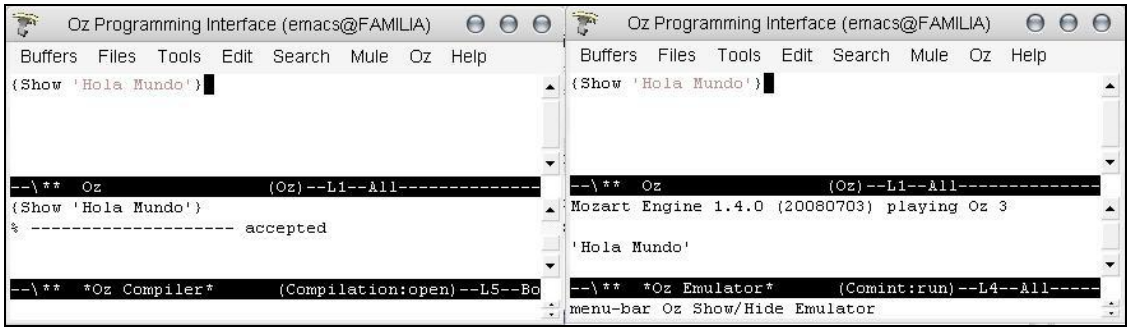

Figura 5.9: Interfaz de Emacs, izquierda compilando y derecha mostrando el resultado.

Un fragmento de código se dice que es *declarativo* sí, siempre que se lo llame con los mismos argumentos y devuelve los mismos resultados con independencia de algún otro estado. Es independiente, sin estado (nada se recuerda entre llamados) y determinístico. Los programas declarativos pueden componerse (probarse modularmente e integrarse) y que su estudio y comprensión se hace siguiendo reglas algebraicas y lógicas simples.

Las variables vistas hasta ahora, que pueden ligarse a un único valor, se conocen como *declarativas*. Desde el punto de vista de la concurrencia, estas variables pueden ya estar ligadas o ligarse en algún futuro. Por eso se las conoce como *dataflow variables*.

Como resultado de este tipo de variables, una *thread* puede ir calculando su resultado en la medida en que las variables que necesite se vayan ligando. La construcción de un *thread* se hace encerrando el código entre *thread ... end*.

```
declare X1 X2 X3
thread
     local Y1 Y2 Y3 in
          {Browse [ Y1 Y2 Y3 ]}
          Y1 = X1+1
          Y2 = X2+Y1
          Y3 = X3+Y2
          {Browse completo}
     end
end
     {Browse X1 X2 X3}
```

Figura 5.10: Ejemplo de thread

En la Figura 5.10 si se alimenta todo el párrafo a Oz, el buffer del Browser muestra [_ _ _] indicando que la thread está a la espera de los valores y [X1 _ _] indicando que el principal también lo está. Al diseño basado en *dataflow* se lo conoce como *dataflow computation.*

5.6 Espacio de Cómputo

Los algoritmos principales de un modelo con restricciones en Mozart se ejecutan en un espacio de cómputo. Este Espacio de Cómputo está compuesto de un almacén de restricciones, propagadores y un distribuidor, que serán explicados a lo largo de este capítulo. En este espacio se hace tanta propagación como sea posible hasta llegar a un estado estable. Si aún no se tiene una solución, o si se quiere otra, se puede hacer distribución creando sub-espacios de cómputo.

La solución del espacio consiste en mapear las variables de números enteros para conseguir satisfacer las restricciones en el almacén de restricciones y todas las restricciones impuestas por los propagadores. A continuación se explicarán los términos introducidos anteriormente [17].

5.7 Propagación

La propagación de restricciones es una regla de inferencia para problemas de dominio finito que reduce los dominios de las variables. La arquitectura computacional para la propagación de restricciones es llamada *espacio* y consiste en un número de propagadores conectados con un almacén de restricciones.

El almacén de restricciones almacena una conjunción de restricciones básicas hasta equivalencia lógica.

Ejemplo: $X \in 4\#9 \wedge Y = 25 \wedge Z \in 12\#20$

Un propagador es un agente concurrente que trata de reducir los dominios de las variables. El propagador impone restricciones no básicas y tiene la capacidad de decir restricciones a un almacén de restricciones. La semántica operacional de un propagador se determina si el propagador puede decir al almacén una restricción básica o no.

En el transcurso de la etapa de propagación, los propagadores pueden tomar distintos estados:

- **Propagador Fallido:** un propagador es inconsistente si no hay asignación de variable que satisface al almacén de restricciones y la restricción impuesta por el propagador, es decir, el propagador es fallido si la semántica operacional que realiza este es inconsistente.

  Ejemplo: $x_1 - x_2 = 10$ sobre $x_1 \in 11\#15$ y $x_2 \in 1\#15$

- **Propagador Implicado:** un propagador es implicado si cada asignación variable satisface el almacén de restricciones también satisface la restricción impuesta por el propagador.

- **Propagador Estable:** Un propagador es estable si es fallido o su semántica operacional no puede decir nueva información al almacén de restricciones.

En el transcurso de la etapa de propagación, el espacio puede tomar distintos estados:

- **Espacio Fallido:** un espacio es fallido si falla uno de sus propagadores.

- **Espacio Estable:** un espacio es estable si todos sus propagadores son estables.

- **Espacio Resuelto:** un espacio está resuelto si no se falla y no hay más propagadores.

Soluciones de un espacio: Una asignación variable es llamada *solución* de un espacio si satisface las restricciones en el almacén de restricciones y todas las restricciones impuestas por los propagadores. Las soluciones de un espacio permanecen invariantes bajo la propagación de restricciones y eliminación de propagadores implicados. [17]

5.7.1 Esquema Operacional

Existen dos esquemas establecidos para la operación de un propagador: la propagación de dominio reduce los dominios de las variables tanto como sea posible y la propagación de intervalo reduce solamente los límites de un dominio.

A modo de ejemplo un propagador para la restricción $3 * X_1 = X_2$ sobre el almacén de restricciones $x_1 \in 1\#15 \;\; x_2 \in 1\#10$:

Bajo propagación de dominio, el propagador puede reducir los dominios a:

$$x_1 \in 1\#3 \;\; x_2 \in \{3,6,9\}$$

Bajo propagación de intervalo, el propagador puede reducir los límites a:

$$x_1 \in 1\#3 \;\; x_2 \in 3\#6$$

En la práctica, la propagación de intervalo es generalmente preferible por sobre la propagación de dominio debido a sus costos computacionales más bajos. [17]

5.7.2 Estado Incompleto de Propagación

La propagación de restricciones no es un método de solución completo, esto debido a que puede suceder que un espacio tenga una única solución y la propagación de restricciones no la encuentre o que el espacio no tenga solución y que la propagación de restricciones no conduzca a un propagador fallido.

Para solucionar este problema se combina la propagación de restricciones y la distribución, ambos lo veremos en el transcurso de este capítulo, proporcionando un método completo de solución para los problemas de dominio infinito. [17]

5.8 Distribución

Para solucionar un problema de dominio finito *P*, se puede elegir una restricción *C* y solucionar siempre $P \cup \{C\}$ y $P \cup \{\sim C\}$, agregar esta nueva restricción no cambia la solución. A lo anterior se dice que se ha distribuido *P* con *C*.

La combinación de la propagación de restricciones y la distribución, proporcionando un método completo de solución para los problemas de dominio infinito.

En un problema, se instala un espacio cuyo almacén contenga las restricciones básicas y cuyos propagadores impongan las restricciones no básicas del problema. Entonces se ejecutan propagadores hasta que el espacio llega a ser estable. Si se falla o se soluciona el espacio, no se hace nada. Si no, se elige una variable aún no determinada x_i y un entero *n*, y se hace dos copias del espacio de cómputo original, agregando un propagador que impone $x_i = n$ a la primera copia y un propagador que impone $x_i \neq n$ a la segunda copia. De esta forma los propagadores pueden volver a actuar en ambos espacios.

La elección de la variable y del valor con el cual se crean la copia del espacio se realiza mediante una Estrategia de Enumeración, la cual está compuesta de heurísticas de selección de variables y heurística de selección de valor. [17]

5.8.1 Estrategias de Enumeración

Antes de continuar definiremos un distribuidor como un agente computacional que implementa estrategias de distribución.

Si un hilo crea un distribuidor se bloquea el hilo hasta que el distribuidor termine su trabajo. Si un paso de distribución es necesario, el distribuidor se activa y genera la restricción con la cual el espacio será distribuido. Los distribuidores se eligen independientemente siempre que sea necesario un paso de distribución. Si esta es necesaria la estrategia selecciona una variable no determinada en la secuencia y distribuye sobre esa variable. Generalmente una estrategia de enumeración se define en una secuencia de variables.

Algunos posibles estándares para distribuir sobre una variable x_i son:

- Distribuir con $x_i = l$, donde *l* es el menor valor posible para x_i .
- Distribuir con $x_i = u$, donde *u* es el mayor valor posible para x_i .
- Distribuir con $x_i = m$, donde *m* es un valor posible para x_i que esté en el medio del menor y el mayor valor posible para x_i .
- Distribuir con $x_i \leq m$, donde *m* es un valor posible para x_i que esté en el medio del menor y el mayor valor posible para x_i (llamado partición de dominio).

Las Estrategias de distribución pueden ser clasificadas en:

- Estrategias de distribución genérica del problema: son estrategias generales que no dependen del problema.

- Estrategias de distribución específicas del problema: son estrategias que dependen de las características del problema para acelerar el proceso de distribución.

En Mozart existen las siguientes estrategias de distribución:

Estrategias de Distribución Ingenua (Native)

Esta estrategia seleccionará la variable indeterminada más a la izquierda de la secuencia y cuanto al valor seleccionará el límite más pequeño del dominio.

Sintaxis: {FD.distribute naive *xv*}, en donde *xv* debe ser un vector de enteros en un dominio finito, por lo tanto, la estrategia seleccionará la variable indeterminada más a la izquierda en *xv*.

Estrategias de Distribución Primer Fallo (First-Fail)

Esta estrategia selecciona la variable indeterminada más a la izquierda para la cual el número de valores diferentes posibles es mínimo.

Sintaxis: {FD.distribute ff *xv*}, en donde *xv* debe ser un vector de enteros en un dominio finito, por lo tanto, la estrategia seleccionará la variable indeterminada más a la izquierda en *xv* cuyo dominio es mínimo.

Estrategias de Distribución Split

Esta estrategia selecciona la variable indeterminada de más a la izquierda en la secuencia cuyo dominio tiene tamaño mínimo, y crea dos puntos de elección con el valor del centro del dominio.

Estrategias de Distribución Genérica

Esta estrategia permite personalizar ciertos parámetros, para crear una estrategia de enumeración que se ajuste a las necesidades de resolución del problema. [17]

Esta estrategia recibe los siguientes parámetros:

- **Order:** que variable se seleccionará

- **Filter:** grupo de variables a considerar

- **Select:** variable a enumerar
- **Value:** que valor se seleccionará
- **Procedure:** se aplica solo cuando se alcanza la estabilidad

A continuación en la Tabla 5.1 se muestra las distintas estrategias de distribución, que en total son 12, las cuales se representaran de la siguiente manera $E_1, E_2, ..., E_{12}$ donde E viene de la palabra estrategia.

	Heurística de Selección		Heurística de Selección de Valores
E_1	DMi	+	MeVal
E_2	DMi	+	MaVal
E_3	DMi	+	MedVal
E_4	DMi	+	MMedVal
E_5	DMa	+	MeVal
E_6	DMa	+	MaVal
E_7	DMa	+	MedVal
E_8	DMa	+	MMedVal
E_9	BOr	+	MeVal
E_{10}	BOr	+	MaVal
E_{11}	BOr	+	MedVal
E_{12}	BOr	+	MMedVal

Tabla 5.1: Constitución de las Estrategias de Distribución

5.9 Árbol de Búsqueda

Alternando fases de propagación y distribución se obtiene un árbol de búsqueda que es siempre finito ya que sus variables están en dominio finito, donde cada nodo del árbol corresponde a un espacio y cada hoja del árbol corresponde a un espacio que ha sido solucionado o fallido.

A cada nivel se aplica propagación al espacio actual, si se falla o se soluciona no se hace nada, de lo contrario, se aplica una fase de distribución, y así se continua hasta construir la solución. [17]

El explorador de Mozart genera un árbol de búsqueda, en donde los espacios seleccionados se representan mediante un rombo verde y los espacios fallidos mediante triángulos rojos. Esto se puede apreciar mejor si observamos la Figura 5.11: Ejemplo de árbol de búsqueda generado por Mozart.

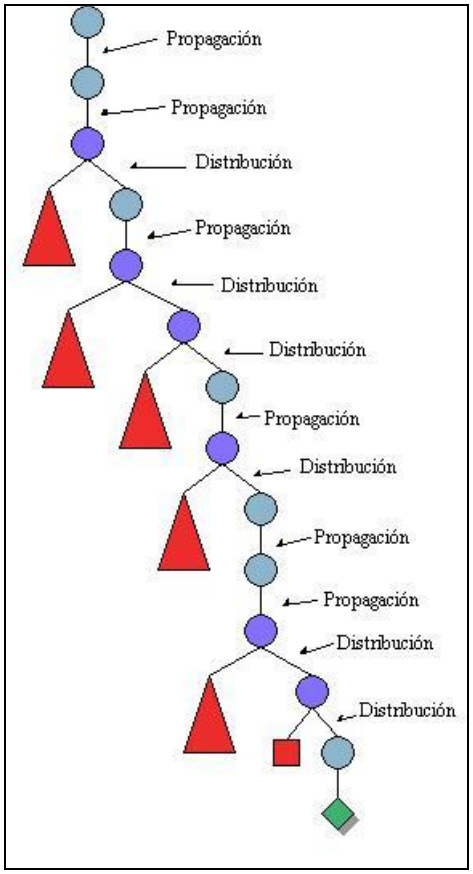

Figura 5.11: Ejemplo de árbol de búsqueda generado por Mozart

5.9.1 Orden de Búsqueda

El orden de exploración de un árbol de búsqueda solo genera un impacto en los recursos de tiempo y de memoria necesitados para encontrar una o todas las soluciones:

- Para encontrar solo una solución, no es necesario explorar el árbol de búsqueda completo. La sentencia *{ExploreOne ScriptP}* comienza la exploración del árbol de búsqueda hasta encontrar la primera solución.

- Para encontrar todas las soluciones, se necesita explorar el árbol de búsqueda por completo, se utiliza la sentencia *{ExploreALL ScriptP}*. Sin embargo si exploramos el árbol de manera Beadth-First (primero en anchura), son exponencialmente más grande con respecto a Depth-First (primero en profundidad).

- Para encontrar la solución optima, se necesita explorar el árbol de búsqueda por completo, se utiliza la sentencia *{ExploreBest ScriptP OrderP}*, la cual sigue una estrategia de Ramificación y Acotamiento (Branch and Bound), la mejor solución se encuentra respecto al procedimiento *OrderP*.

El motor de búsqueda explora los árboles de búsqueda de la manera depth-first y cuando el motor distribuye con una restricción C, explora el espacio obtenido con C primero y el espacio obtenido con ~C segundo, provocando que la exploración de un árbol de búsqueda sea un proceso determinista, que proviene de una estrategia de enumeración determinista que genera las restricciones para distribuir. [17]

Capítulo 6

Ejemplos Clásicos de CSP

6.1 Problema de Coloración

El problema del Coloración es un problema clásico que se puede formular como un CSP. Un ejemplo del problema de coloración es el problema de coloración de mapas donde hay un conjunto de colores y se desea colorear cada región del mapa de manera que las regiones adyacentes tengan distintos colores. [1] [3][4] En la formulación del CSP, hay una variable por cada región del mapa, y el dominio de cada variable es el conjunto de colores disponible. Para cada par de regiones contiguas existe una restricción sobre las variables correspondientes que no permite la asignación de idénticos valores a las variables. Este mapa puede ser representado mediante un grafo donde los nodos son variables que representan a los colores asociados a las regiones y cada par de regiones adyacentes están unidas por una arista.

6.1.1 Modelo del problema como un CSP

El problema de la coloración de un grafo consiste en decidir si los nodos del grafo pueden colorearse empleando k colores de manera que dos nodos adyacentes no estén coloreados con el mismo color. Su representación como un CSP se puede definir de la siguiente forma:

- X es el conjunto de nodos.
- D asigna el dominio {1, 2,…, k} a cada variable, donde cada elemento del dominio denota uno de los posibles colores.
- R contiene la restricción $X_i \neq X_j$ para cada par $\left(X_i, X_j\right)$ de nodos adyacentes

6.1.1.1 Ejemplo Clásico

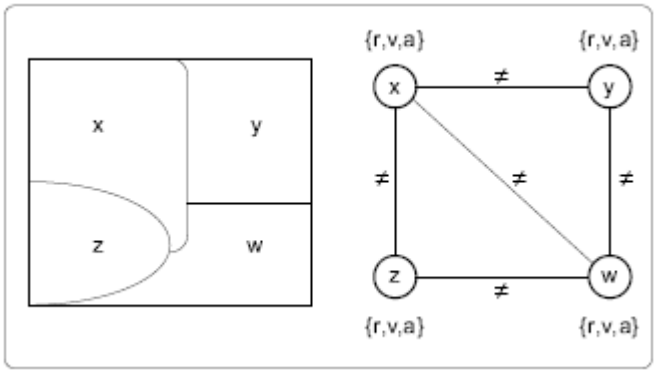

Figura 6.1: Problema de coloración del mapa

En la Figura 6.1, tenemos un mapa con cuatro regiones x, y, z, w para ser coloreadas con los posibles colores Rojo, Verde, Azul. La formulación CSP sería:

- Variables: $\{x, y, z, w\}$
- Dominio: $D_w = D_x = D_y = D_z = \{r, v, a\}$, donde r representa rojo, v representa verde y a representa azul
- Restricciones: $C = \{C_{x,w}, C_{x,y}, C_{x,z}, C_{z,w}, C_{y,w}\}$,
  Es decir, $\{x \neq y, x \neq w, x \neq z, y \neq w, z \neq w\}$

Un CSP binario, puede ser representado mediante una red de restricciones, donde los nodos representan las variables y los arcos representan las restricciones entre las mismas. En la Figura 6.1 en la parte derecha, se representa la red correspondiente al problema del ejemplo, donde las variables correspondientes a regiones adyacentes están conectadas por una arista. Hay cinco restricciones en el problema, es decir, cinco aristas en la red. Una solución para el problema es la asignación (x, r), (y, v), (z, v), (w, a). En esta asignación, todas las variables adyacentes tienen valores diferentes.

6.1.1.2 Ejemplo del mapa de las provincias de la 5ª región

A continuación se modelará el problema de coloración de mapas con el mapa de las provincias de la Región de Valparaíso de Chile, que se muestra en la Figura 6.2 en donde Z: San Antonio, X: Valparaíso, Y: Quillota, W: Los Andes, V: San Felipe de Aconcagua y U: Petorca.

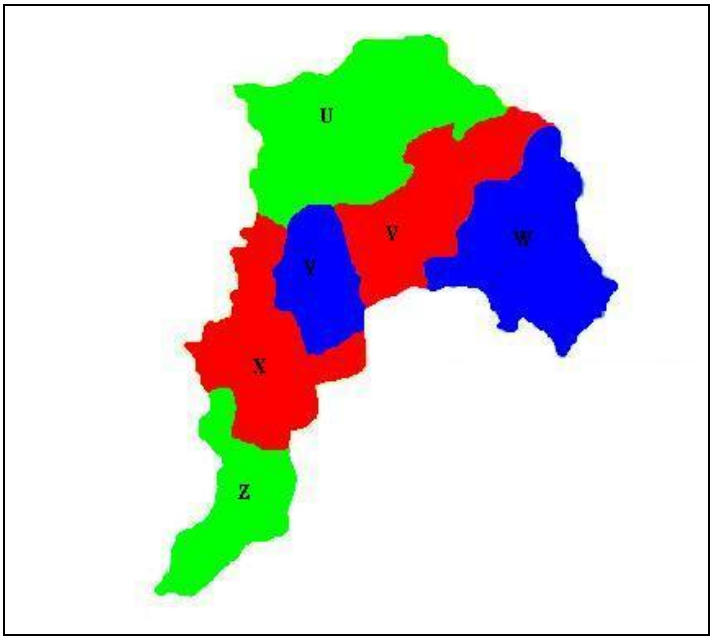

Figura 6.2: Mapa de la 5ª Región en donde.

La modelación del CSP es:

- Variables: $\{u, v, x, y, z, w\}$
- Dominio: $D_u = D_v = D_w = D_x = D_y = D_z = \{r, v, a\}$, donde r representa rojo, v representa verde y a representa azul.
- Restricciones: $C = \{C_{y,v}, C_{y,x}, C_{y,u}, C_{v,w}, C_{v,u}, C_{x,z}, C_{x,u}\}$,
  Es decir, $\{y \neq v, y \neq x, y \neq u, v \neq w, v \neq u, x \neq z, x \neq u\}$

La resolución de este problema se encuentra adjuntada en el Anexo: Código del Problema de Coloración de Mapas en O .

6.2 Problema de N-Reinas

En esta sección, vamos a formular el problema de N-reinas de acuerdo con la definición oficial del CSP, y poniendo en manifiesto que un problema puede formularse como un CSP de diferentes maneras.

El Problema de las *n*-Reinas fue propuesto originalmente para *n* = 8 en el año 1848 en un trabajo anónimo publicado en Berliner Schachgesellschaft, siendo posteriormente atribuido a Max Bezzel. Sin embargo, la publicación detallada más antigua que se registra fue realizada por Nauck en 1850 [27]. En ese mismo año, Gauss postuló la existencia de 72 soluciones para *n* = 8, posteriormente en el año 1874 Glaisher probó la existencia de 92 soluciones [28]. Muchas de las soluciones planteadas se basan en proporcionar una fórmula específica para colocar las reinas o extrapolar conjuntos pequeños de soluciones para proporcionar soluciones para valores de *n* más grandes. Luego se ha demostrado que el número de soluciones crece en forma exponencialmente a medida que se va incrementando *n.*

Definición: consiste en encontrar una distribución de *n* reinas en un tablero de ajedrez de dimensiones *n x n,* de forma que éstas no se ataquen.

- Formulación: 1 reina por fila
- Variables: reinas, X_i reina en la fila i-ésima
- Dominios: Columnas Posibles {1, 2,…, n}
- Restricciones: no colocar dos reinas en:
  - La misma columna
  - La misma diagonal
  - La misma fila

Este problema tiene 2 versiones, la más simple consiste en encontrar exactamente una solución válida para un valor *n* dado. La otra versión, más difícil, consiste en encontrar todas las soluciones posibles para un valor *n*.

Se puede observar que este problema tiene una solución $\left(Q(1)=1\right)$ para *n* = 1, no tiene solución para *n* = 2 y *n* = 3 y tiene 2 soluciones para *n* = 4. En la Tabla 6.1 se muestra el número total de Soluciones $Q(n)$ para $4 \le n \le 20$. [29]

n	$Q(n)$
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2.680
12	14.200
13	73.712
14	365.596
15	2.279.184
16	14.772.512
17	95.815.104
18	666.090.624
19	4.968.057.848
20	39.029.188.884

Tabla 6.1: Número total de soluciones $Q(n)$ para $4 \leq n \leq 20$

Para solucionar este problema, se han diseñado numerosos algoritmos como Backtracking, Algoritmos Genéticos, Búsqueda local con resolución de conflicto, programación entera y redes neuronales, entre otros. El problema de n-reinas pertenece a la clase de problema NP-duros [30], pero se resuelve fácilmente en tiempo polinomial cuando solamente se busca una solución [31].

El Backtracking, en general es ineficiente y para el problema de las n-reinas no es fácil encontrar soluciones para $n > 100$ en tiempos razonables [32], sin embargo, Kalé [33] diseño un algoritmo especializado de backtracking que consigue resolver el problema hasta tamaños de orden de 1000.

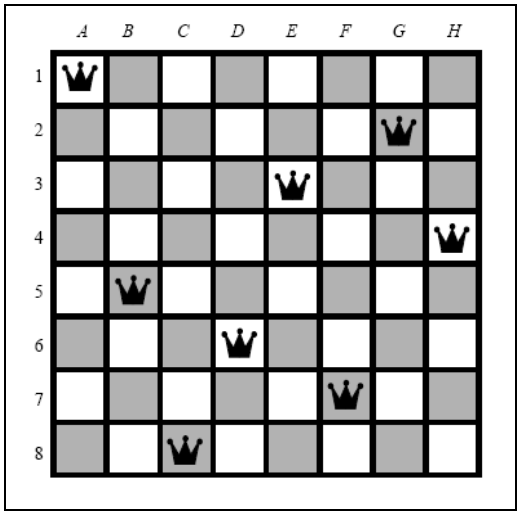

Figura 6.3: Una posible solución al problema de 8-reinas.

Una manera de modelar el problema de 8-reinas como un problema CSP es verlo como un problema con ocho variables, es decir un conjunto finito de variables, donde cada uno de los cuales pueden tener un valor de la A a la H. La tarea consiste en asignar valores a las variables que cumplan las restricciones antes especificadas.

6.2.1 Ejemplo del problema de N-reinas

Para entender mejor, se analizará el problema de las n-reinas para $n = 6$. Esto consiste en ubicar 6 reinas en un tablero de 6x6 sin que ellas se ataquen o entren en conflicto. Dichas soluciones se presentan en los gráficos de la Figura 6.4. Estas soluciones fueron encontradas utilizando el modelo que se explica en la sección 6.2.2.

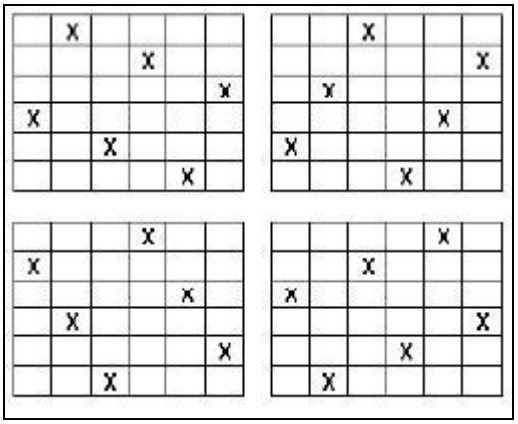

Figura 6.4: Soluciones al Problema de las 6-reinas

6.2.2 Modelo del problema de 6-reinas como un CSP

Para llegar a la solución, se modeló el problema teniendo presente lo siguiente:

- La Función Objetivo no tiene un uso práctico ya que se sabe previamente el valor de *n*. Así, el problema no necesita ser presentado como un CSOP, sino simplemente como un CSP.
- Dos reinas no pueden estar en la misma Fila.
- Dos reinas no pueden estar en la misma Columna.
- Dos reinas no pueden estar en la misma Diagonal.

Dado lo anterior tenemos el siguiente modelo:

Variables: $X_{i,j} = 1$ si existe una reina en la posición (i, j) y 0 en lo contrario.

Dominio: {0, 1}

Restricciones:

Restricción en las Filas $$\sum_{j=1}^{n} X_{ij} = 1 \qquad ; \forall i = \{1,...,n\} \tag{6.1}$$

Restricción en las Columnas $$\sum_{i=1}^{n} X_{ij} = 1 \qquad ; \forall j = \{1,...,n\} \tag{6.2}$$

Restricción en las Diagonales 1 $$\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n}}_{i+j=k} X_{ij} \le 1 \qquad ; \forall k = \{3,...,2n-1\} \tag{6.3}$$

Restricción en las Diagonales 2 $$\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n}}_{i-j=k} X_{ij} \le 1 \qquad ; \forall k = \{1-n,...,n-1\} \tag{6.4}$$

Así, si n = 6, tendremos las variables $X_{11}, X_{12}, ..., X_{66}$, correspondientes a las 36 casillas del Tablero.

Ahora bien, $\sum_{j=1}^{n} X_{ij} = 1 \; ; \forall i = \{1,...,n\}$, significa que en una fila determinada tiene que haber 1 única reina. De la misma forma, $\sum_{i=1}^{n} X_{ij} = 1 \; ; \forall j = \{1,...,n\}$, significa que en una columna determinada tiene que haber 1 única reina.

Finalmente, $\underbrace{\sum_{i=1}^{n} \sum_{j=1}^{n}}_{i+j=k} X_{ij} \leq 1 \; ; \forall k = \{3,...,2n-1\}$ y $\underbrace{\sum_{i=1}^{n} \sum_{j=1}^{n}}_{i-j=k} X_{ij} \leq 1 \; ; \forall k = \{1-n,...,n-1\}$ nos aseguran que en una diagonal, a lo más habrá 1 reina.

6.2.3 Modelo del problema de 8-reinas como un CSP

Para formalizar un problema como un CSP, es necesario identificar un conjunto de variables, un conjunto de dominios y un conjunto de restricciones. Una manera de formalizar el problema de 8-reinas como un CSP es hacer de cada una de las ocho filas del problema de 8-reinas una variable: el conjunto de las variables $Z = \{Q_1, Q_2, ..., Q_8\}$. Cada una de estas ocho variables puede tomar una de las ocho columnas como su valor. Si la etiqueta las columnas con valores de 1 al 8, para efectos de cálculo que se hará evidente más adelante, entonces los dominios de todas las variables en este CSP son las siguientes:

$$D_{Q_1} = D_{Q_2} = ... = D_{Q_8} = \{1,2,3,4,5,6,7,8\}$$

Si se ve la serie de restricciones, el hecho de representar a cada fila como una variable, nos asegura de que no puedan estar dos reinas en la misma fila. Para asegurar esto tenemos la siguiente restricción:

Restricción (1) = $\forall i, j : Q_i \neq Q_j$

Para asegurar de que no puedan estar dos reinas en la misma diagonal, se incluye la siguiente restricción en la serie de restricciones:

Restricción (2) = $\forall i, j$, si $Q_j = b$, donde $i - j \neq a - b$, y $i - j \neq b - a$

Representar estas restricciones, se podría registrar explícitamente el conjunto de todos los valores compatibles entre cada par de variables. Alternativamente, se puede hacer que las funciones o procedimientos retorne verdadero o falso, dependiendo de si las etiquetas son compatibles o no.

En virtud de este problema de la formalización, hay 8^8 combinaciones de valores para las ocho variables que deben ser consideradas. En general, un problema de N-reinas existen N^N soluciones candidatas a ser consideradas. [25]

En el presente trabajo se utilizo este modelo para resolver el problema en Oz con CSP, la resolución de este problema se encuentra adjuntada en el Anexo: Código del problema de N-Reinas en O.

6.2.3.1 Modelo alternativa del problema de 8-Reinas

Hay que tener en cuenta que a menudo hay más de una manera de formalizar un problema como un CSP. El problema de N-reinas no tiene que ser formalizado en la forma anterior, una alternativa es utilizar la representación $Q_1, Q_2, ..., Q_8$ para representar las posiciones de las reinas, en vez de las columnas de cada reina en la formalización. Si las 64 casillas de 8x8 en el tablero están numerados del 1 al 64, entonces el dominio de cada variable se convierte en $\{1, 2, 3, ..., 64\}$. En otras palabras:

$$Z = \{Q_1, Q_2, ..., Q_8\}$$
$$D_{Q_1} = D_{Q_2} = ... = D_{Q_8} = \{1, 2, 3, 4, ..., 64\}$$

Si se supone que el número de plazas es de izquierda a derecha y de arriba a abajo. Entonces un número que representa una casilla, la fila y la columna de la casilla puede ser calculado de la siguiente manera:

Fila = (numero dividido por 8) + 1
Columna = (numero modulo 8) + 1

Contando las filas y las columnas de dos casillas, se puede comprobar si son compatibles entre sí utilizando una función que compare, un seudo-código de esto sería:

```
compatible(N1, N2);

Fila1 = (N1 div 8) + 1
Columna1 = (N1 mod 8) + 1
Fila2 = (N1 div 8) + 1
Columna2 = (N2 mod 8) + 1
R1 ≠ R2
Columna1 ≠ Columna2
Fila1 – Fila2 ≠ Columna1 – Columna2
Fila1 – Fila2 ≠ Columna2 – Columna1
```

Figura 6.5: Seudo-Código de la comprobación

Algunas formulaciones son más fáciles de resolver que otras, el problema de 8-reinas formalizado en esta sección permite 64^8 combinaciones posibles, lo que hace que el

problema potencialmente más difíciles de resolver que el CSP formalizado en la sección anterior (que sólo permite 8^8 combinaciones posibles). La formalización en la sección anterior, de hecho, se ha construido en la restricción de que no hay dos reinas se puede colocar en la misma fila. [25]

Capítulo 7

Prueba de Rendimiento

7.1 Introducción

En la presente sección se presenta la implementación en el sistema de desarrollo Mozart de los problemas de N-reinas y Coloración de Mapas explicados en el capitulo anterior, dichas pruebas han sido utilizadas para ilustrar el efecto que tienen las heurísticas de selección de variables y de valor a la hora de ser aplicadas en el problema para encontrar su resolución. A continuación describiremos como se representa dichas estrategias en nomenclatura OZ.

Para las estrategias que utilizan heurísticas de selección de variable del tipo DMi, el parámetro para *order* en la distribución genérica se fija en *size*, el cual permite la selección de variable cuyo tamaño de dominio sea mínimo, lo que respecta al parámetro *value* este puede ser fijado en *min*, *max* o *mid* para heurística de valor del tipo Metal, Maval y Medial respectivamente.

```
{FD.distribute generic(order: size value: min)Sol}
```

Figura 7.1: Representación de la estrategia constituida por las heurísticas DMi y MeVal

Para las estrategias constituidas por la heurística de selección de valor del tipo MMedVal el parámetro *value* debe ser fijado a una función que retorne aquel valor del dominio que es inmediatamente mayor al valor medio, en la Figura 7.2 se muestra su implementación.

```
{FD.distribute generic(order: size
                       value: fun{$ S}
                          {FD.reflect.nextLarger
S{FD.reflect.mid S}}
                        end) Sol }
```

Figura 7.2: Implementación de heurísticas del tipo MMedVal

Del mismo modo que se deben fijar los parámetros para *order* y *value*, para la estrategias que utilizan heurísticas del tipo DMa pero en lugar de fijar *order* a *size* como se haría en el caso DMi, se fija una función que nos permite discriminar entre el tamaño de los dominios de la variables y elegir aquella que tenga el tamaño del dominio mayor, dicha

función se muestra en la Figura 7.3 ,con respecto al parámetro *value* se aplica de la misma forma que en las estrategias que utilizan heurísticas de selección de variable del tipo DMi.

```
{FD.distribute generic(order: fun {$ X Y}

                        if {FD.reflect.size X} >
{FD.reflect.size Y} then

                         true

                                          else

                         false

                         end

                      end

                      value: min) Sol}
```

Figura 7.3: Implementación de estrategia constituida por heurísticas DMa y MeVal

Todo lo explicado anteriormente con respecto al parámetro *value* en las heurísticas del tipo DMi y DMa se conservan para las estrategias que utilizan heurísticas de selección de variables estáticas (BOr), sin embargo el parámetro *order* se fija a *naive,* el cual selecciona la variable de más a la izquierda en una lista que establece el orden de selección, en este caso particular la denominaremos como "Lista".

```
{FD.distribute generic(order: naive

                    value: min) {Map Lista fun{$ I} Sol.I
end}}
```

Figura 7.4: Ejemplo de implementación de una estrategia que utiliza heurística BOr y MeVal

Ver la Tabla 5.1: Constitución de las Estrategias de Distribución para ver las distintas estrategias de distribución y como están formadas.

7.2 Consideraciones Previas a la Simulación Computacional

Para la implementación de los script mostrados en el anexo del presente trabajo se utiliza la plataforma de desarrollo Mozart, la versión utilizada de la plataforma de desarrollo corresponde a la versión 1.4.0 y las pruebas realizadas se ejecutaron en un computador con procesador AMD Dual Core 2 de 5200 con 2 GB de memoria Ram.

Los indicadores de Desempeños aplicados en la presente sección, corresponde a evaluar el rendimiento de los problemas de N-Reinas y coloración de mapas bajo la plataforma de desarrollo Mozart, problemas en el capítulo 7, frente a las diversas Estrategias de Distribución. Con el objetivo de medir y establecer tales diferencias es necesario establecer algún indicador de desempeño que permita reflejar con claridad el

efecto de aplicación de una estrategia de enumeración u otra, las cuales se muestran en la Tabla 5.1, por lo tanto se establecieron como indicadores de desempeño y eficiencia los siguientes:

- Tiempo (T): mide el tiempo requerido para resolver el problema.
- Número de Nodos (N): cuenta la cantidad de nodos o espacios generados para encontrar la solución del problema, incluyendo los nodos que conducen a una solución y las malas que obligan a retroceder.
- Número de Fallos (F): cuenta la cantidad de fallos producidos antes de encontrar la primera solución o todas las soluciones.
- Profundidad (P): indica la profundidad el árbol binario generado por Mozart/OZ.

Cuando ocurra un error del programa o falla su ejecución debido a la aplicación de heurísticas que no soportan el modelo del problema, incluso provocando que la plataforma Mozart se caiga o ocupe el 100% del procesador y/o ocupe más de 2 GB de memoria RAM causando el bloqueo del programa, se anotaran estos sucesos con el símbolo "-" en las tablas de resultados que se muestran a lo largo de estos capítulos.

7.3 Búsqueda de la Primera Solución del Problema N-Reinas

Tabla 7.1: Resultado de la prueba de rendimiento para 4-Reinas, 8-Reinas y 20-Reinas

	4-Reinas				8-Reinas				20-Reinas			
	(F)	(N)	(T)	(P)	(F)	(N)	(T)	(P)	(F)	(N)	(T)	(P)
E1	2	3	0	3	23	25	0	9	33	43	0	19
E2	2	3	0	3	23	25	0	9	33	43	0	19
E3	0	1	0	2	2	5	0	6	15	26	0	18
E4	0	1	0	2	2	5	0	6	38	51	0	18
E5	2	4	0	4	127	131	0	19	325370	325382	13.68	83
E6	2	4	0	4	127	131	0	19	325370	325382	14.61	83
E7	0	1	0	2	0	3	0	4	126484	126495	6.04	78
E8	0	1	0	2	8	12	0	9	2501	2517	93	46
E9	2	3	0	3	24	26	0	9	37320	37330	1.15	39
E10	2	3	0	3	24	26	0	9	37320	37330	1.28	39
E11	0	1	0	2	2	5	0	6	487	502	15	26
E12	0	1	0	2	2	5	0	6	13	27	0	20

En la Tabla 7.1 y la Tabla 7.2 se puede apreciar que las heurísticas con mejor rendimiento con respecto a los criterios utilizados en estas pruebas de rendimiento, visto en la sección 7.2, son las estrategias E3 y E4 en respecto a que soporta todo el intervalo de muestras, es decir, los problema de 4, 8, 20, 50, 100 reinas respectivamente, en cambio si solo nos limitamos a los problema con un número de reinas menores de 50, la heurística

E12 tiene un optimo desempeño, pero no soporta problemas de este tipo con un número de reina mayores e igual de 50 reinas.

Tabla 7.2: Resultado de la prueba de rendimiento para 50-Reinas y 100-Reinas

	50-Reinas				100-Reinas			
	(F)	(N)	(T)	(P)	(F)	(N)	(T)	(P)
E1	512	553	16	50	22	115	16	97
E2	512	553	16	50	22	115	0	97
E3	7	49	15	46	7	103	0	98
E4	1	46	0	47	2	98	32	98
E5	-	-	-	-	-	-	-	-
E6	-	-	-	-	-	-	-	-
E7	-	-	-	-	-	-	-	-
E8	-	-	-	-	-	-	-	-
E9	-	-	-	-	-	-	-	-
E10	-	-	-	-	-	-	-	-
E11	-	-	-	-	-	-	-	-
E12	-	-	-	-	-	-	-	-

La resolución de este problema se encuentra adjuntada en el Anexo: Código del problema de N-Reinas en Oz.

7.4 Búsqueda de Solución del Problema Coloración de Mapas

Tabla 7.3: Resultado de prueba de rendimiento para la Primera Solución de problema de Coloración de Mapas

	Primera Solución							
	4 Colores				5 Colores			
	(F)	(N)	(T)	(P)	(F)	(N)	(T)	(P)
E1	16	21	0	10	16	22	0	11
E2	0	6	0	7	0	7	0	8
E3	16	21	0	10	16	22	0	11
E4	12	18	0	8	12	18	0	8
E5	36	41	0	13	32	42	0	13
E6	0	6	0	7	0	7	0	8
E7	36	41	0	12	36	42	0	12
E8	32	28	0	10	32	38	15	10
E9	16	21	0	10	16	22	0	11
E10	0	6	0	7	0	7	0	8
E11	16	21	0	10	16	22	0	11
E12	12	18	0	8	12	18	0	8

Con respecta al problema de coloración de mapas, la Tabla 7.3 y la Tabla 7.4 se demuestra que no hay diferencia radicales entre las estrategia con respecto al problema de

n-reinas, debido a la complejidad de esta. Por lo tanto en esta prueba de rendimiento de heurística, las heurísticas que obtuvieron mejor rendimiento con respecto a los criterios explicado en la sección anterior son las heurísticas E2, E6 y E10, quedando demostrado que las heurística que en Selección de variable sea del tipo DMa son las que tienen peor rendimiento en este tipo de problema.

Tabla 7.4: Resultado de prueba de rendimiento para Todas las Soluciones de problema de Coloración de Mapas

	Todas las Soluciones							
	4 Colores				**5 Colores**			
	(F)	**(N)**	**(T)**	**(P)**	**(F)**	**(N)**	**(T)**	**(P)**
E1	16	159	0	14	16	1599	46	21
E2	16	159	0	11	16	1599	47	17
E3	16	159	0	14	16	1599	126	21
E4	16	159	16	12	16	1599	139	19
E5	186	329	15	19	602	2185	171	26
E6	186	329	0	16	602	2185	156	22
E7	186	329	31	19	602	2185	78	26
E8	186	329	32	17	602	2185	63	24
E9	16	159	0	15	16	1599	62	22
E10	16	159	15	12	16	1599	16	18
E11	16	159	15	15	16	1599	78	22
E12	16	159	16	13	16	1599	127	20

La resolución de este problema se encuentra adjuntada en el Anexo: Código del Problema de Coloración de Mapas en Oz.

Capítulo 8

Prototipo del Sistema

8.1 Modelo del TSP como un CSOP y su adaptación a Oz

Como se ha visto en el Capítulo 5, en pocas palabras el CSP se resume de la siguiente manera: $CSOP = \langle X, D, C, f \rangle = \langle CSP, f \rangle$. Entonces el modelamiento del TSP como un CSOP quedaría de la siguiente forma:

***X* (Variables):**

Variable de decisión: $X_{i,j} \begin{cases} 1 \text{ si el vendedor va desde la ciudad } i \text{ a la ciudad } j \\ 0 \text{ en otro caso} \end{cases}$

Desde cada ciudad $i \in N$ se puede viajar hasta cada ciudad $j \in N \setminus \{i\}$.

Donde *n* es la cantidad de ciudades distintas que a la cual debe viajar el vendedor, las que se encuentran indexadas al través del conjunto $N = \{1,...,n\}$. $C_{i,j}$ es el costo total de distancia entre las ciudades i, j que el vendedor va a recorrer.

***D* (Dominio):** $x_{ij} \in \{0,1\} \quad \forall i, j \in N$

***C* (Restricciones):**

$$\sum_{i=1}^{i=N} x_{ij} = 1 \qquad (j = 1...N) \tag{6.5}$$

$$\sum_{j=1}^{j=N} x_{ij} = 1 \qquad (i = 1...N) \tag{6.6}$$

Son aplicadas implícitamente en Oz mediante la declaración de {FD.distinct Ruta}, donde Ruta es el vector de tuplas que indican el camino escogido por el vendedor en el problema del TSP, es decir, la declaración FD.distinct es una restricción que obliga que cada elemento del vector del tuplas sea distinto, esto automáticamente restringe que el vendedor visite solo una vez cada ciudad, impidiendo ciclos internos logrando satisfacer las restricciones indicadas anteriormente y a la vez esto satisface las siguientes restricciones:

La restricción que indica que sólo se debe visitar 1 vez cada ciudad (a) y la restricción que indica que se debe comenzar en cada ciudad (b).

Para restringir que el vendedor empiece en la ciudad de origen escogida por el cliente y/o usuario se utiliza la siguiente declaración: Ruta.Cantidad =: 1 % Crea una restricción que nos asegura que la última ciudad visitada sea la ciudad de origen elijada.

Rutas.1 =: 1 % Crea una restricción que nos asegura que la primera ciudad visitada sea la ciudad de origen elijada, este es el vector final utilizado para generar la solución del TSP.

```
{For 1 Cantidad 1
     proc{$A}

     Rutas.(A+1)=:Ruta.A
end}
```

Crea una restricción que nos asegura que las 2ª ciudad visitada hasta la N-ésima del vector de Tuplas Rutas, sus componentes seas del vector de duplas Rata, logrando que el Rutas, su primera y última ciudad sea las ciudad de origen.

Un ciclo, es aquel camino que satisface las condiciones, es decir, pasa por todas las ciudades al menos una vez y sólo una vez, comenzando y terminando en la misma ciudad. También tenemos que un ciclo interno (o subciclo), comienza y termina en la misma ciudad, pero no pasa por todas las ciudades.

***f* (Función objetivo a minimizar):** $Mín\ z = \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$ indica el costo total (distancia), esta función objetivo se logra con el fragmento de código:

```
Suma_Costos     = {FD.decl} = {FD.sum Distancias '=:'} con
esto de logra la sumatoria

Lo siguiente genera la solución:

    {FD.distribute generic(order:size  value:max) Ruta}

{For 1 Cantidad+1 1
  proc {$ A}
     Nombre_Ciudades.A = Vector_Ciudades_modificada.(Rutas.A)
  end}

   {For 1 Cantidad 1
  proc {$ A}
     Distancias.A = Matriz_Nueva.(Rutas.A).(Rutas.(A+1))
  end}
```

8.2 Matrices de Distancias Utilizadas

Tabla 8.1: Matriz de Distancia de las Ciudades de Chile (A)

	Angol	Antofagasta	Arica	Calama	Cartagena	Castro	Cauquenes	Concepción	Coyhaique	Constitución	Copiapó	Coquimbo	Curico	Chillán	Illapel	Iquique	La Serena	La Unión	Lebu	Linares	Los Andes	Los Ángeles	Los Lagos	Osorno	Ovalle	Pucón	Puerto Montt	Puerto Varas	La Ligua	Punta Arenas	Rancagua	San Antonio	San Felipe	San Fernando	Santiago	Talca	Talcahuano	Temuco	Tocopilla	Valdivia	Valparaíso	Vallenar	Villa Alemana	Villarica	Viña del Mar	Zapallar
Angol		1946	2638	2143	594	649	283	147	1241	374	1372	1027	378	166	849	2358	1038	351	161	270	646	**60**	288	378	972	237	480	464	719	2571	483	585	655	429	568	316	161	131	2129	297	674	1231	673	211	683	733
Antofagasta	1946		701	215	1391	2552	1730	1880	3011	1736	566	918	1571	1768	1184	492	887	2272	2020	1684	1337	1878	2210	2274	992	2158	2377	2385	1234	4451	1448	1397	1321	1516	1361	1618	1884	2038	185	2202	1313	722	1313	2132	1319	1250
Arica	2638	**701**		**593**	2084	3253	2423	2581	3712	2428	1261	1611	2263	2489	1876	**316**	1602	2965	2713	2376	2030	2579	2902	2975	1685	2850	3078	3078	1926	5152	2149	2030	2014	2209	2062	2319	2577	2739	536	2903	2020	1414	2006	2825	2011	1943
Calama	594	215	593		1589	2767	1927	2074	3360	1933	771	1116	1768	1978	1381	380	1106	2469	2217	1881	1534	2087	2407	2497	1189	2355	2598	2582	1431	4689	1657	1594	1518	1713	1574	1831	2081	2249	156	2415	1525	**919**	1510	2329	1516	1448
Cartagena	584	1391	2084	1589		1218	379	525	1811	384	818	472	219	429	295	1804	484	921	668	332	178	539	858	948	417	806	1050	1033	165	3140	136	**9**	177	165	112	282	533	701	1575	866	**79**	677	**78**	780	**88**	148
Castro	649	2552	3253	2767	1218		1218	**801**	*414	999	1966	1651	1002	776	1474	3036	1664	314	778	**894**	1270	676	**361**	**284**	1596	506	175	190	1343	1994	1104	1209	1279	1054	1191	934	**805**	514	2753	385	1310	1855	1297	**480**	1308	1358
Cauquenes	283	1730	2423	1927	379	1218		**140**	1500	105	1157	811	162	118	634	2143	823	610	283	**98**	431	228	547	637	756	495	739	722	503	2829	268	369	439	214	353	100	147	390	1913	555	459	1015	457	469	468	518
Concepción	147	1880	2581	2074	525	801	140		1260	245	1314	958	309	112	781	2372	993	493	143	201	577	127	430	523	903	378	626	606	650	2700	432	516	586	361	519	262	**16**	267	2060	452	639	1162	604	353	615	665
Coyhaique	1241	3011	3712	3360	1811	*414	1500	1260		1591	2451	2244	1595	1384	2067	3503	2255	**907**	1370	1487	1863	1275	**954**	**869**	2189	1098	**764**	**782**	1936	1434	1701	1802	1872	1646	1786	1533	1397	1112	3346	**973**	1891	2448	1890	1073	1900	1950
Constitución	374	1736	2428	1933	**384**	**999**	**105**	**245**	1591		1162	**817**	**168**	**209**	**639**	2148	**828**	**701**	**388**	**112**	**436**	**319**	**638**	**728**	**762**	**587**	**830**	**814**	**509**	2921	**273**	**375**	**445**	**219**	**358**	**106**	**253**	**481**	1919	**647**	**464**	1021	**463**	**561**	**473**	**523**
Copiapó	1372	**566**	1261	**771**	**818**	1992	1157	1314	2451	1162		**345**	**997**	1207	**610**	**986**	**336**	1699	1447	1110	**764**	1317	1636	1726	**419**	1584	1828	1812	**660**	3919	**887**	**824**	**748**	**943**	**803**	1060	1311	1479	**757**	1644	**754**	**148**	**740**	1559	**745**	**677**
Coquimbo	1027	**918**	1611	1116	**472**	1651	**811**	**958**	2244	**817**	**345**		**652**	**862**	**265**	1331	**11**	1353	1101	**765**	**418**	**971**	1291	1381	**94**	1239	1482	1466	**315**	3573	**514**	**478**	**402**	**597**	**458**	**715**	**965**	1133	1102	1299	**409**	**204**	**394**	1213	**400**	**332**
Curicó	**378**	1571	2263	1768	**219**	1002	**162**	**309**	1595	**168**	**997**	**652**		**213**	**475**	1983	**664**	**704**	**443**	**116**	**271**	**322**	**642**	**732**	**597**	**590**	**833**	**817**	**344**	2924	**109**	**210**	**280**	**55**	**194**	**66**	**316**	**484**	1754	**650**	**300**	**856**	**298**	**564**	**309**	**359**
Chillán	**166**	1768	2489	1978	**429**	**776**	**118**	**112**	1384	**209**	1207	**862**	**213**		**684**	2193	**873**	**493**	**241**	**105**	**481**	**111**	**430**	**521**	**807**	**379**	**622**	**606**	**554**	2713	**318**	**420**	**490**	**264**	**403**	**151**	**105**	**273**	1964	**439**	**509**	**1066**	**508**	**353**	**518**	**568**
Illapel	**849**	1184	1876	1381	**295**	1474	**634**	**781**	2067	**639**	**610**	**265**	**475**	**684**		1596	**276**	1176	**924**	**587**	**241**	**794**	1113	1204	**184**	1062	1305	1289	**137**	3396	**364**	**301**	**225**	**420**	**280**	**538**	**788**	**956**	1367	1122	**232**	**469**	**217**	1036	**223**	**154**
Iquique	2358	**492**	**316**	**380**	1804	3036	2143	2372	3503	2148	**986**	1331	1983	2193	1596		1322	2685	2432	2096	1750	2303	2622	2712	1405	2570	2814	2798	1646	4905	1873	1809	1734	1929	1798	2046	2297	2465	**229**	2630	1740	1134	1726	2545	1731	1663
La Serena	1038	**887**	1602	1106	**484**	1664	**823**	**993**	2255	**828**	**336**	**11**	**664**	**873**	**276**	1322		1365	1113	**776**	**430**	**983**	1302	1393	**85**	1251	1494	1487	**326**	3585	**553**	**490**	**414**	**609**	**469**	**726**	**977**	1145	1092	1311	**421**	**194**	**406**	1225	**412**	**343**
La Unión	351	2272	2965	2469	**921**	**314**	**610**	**493**	**907**	**701**	1699	1353	**704**	**493**	1176	2685	1365		**480**	**597**	**973**	**385**	**64**	**44**	1298	**208**	**145**	**129**	1045	2236	**810**	**911**	**981**	**756**	**895**	**643**	**507**	**221**	2456	**66**	1001	1558	**999**	**182**	1010	1060
Lebu	**161**	2020	2713	2217	**668**	**778**	**283**	**143**	1370	**388**	**1447**	1101	**443**	**241**	**924**	2432	1113	**480**		**344**	**720**	**216**	**417**	**507**	1046	**365**	**609**	**593**	**793**	2700	**558**	**659**	**729**	**504**	**643**	**390**	**155**	**260**	2203	**425**	**749**	1305	**747**	**340**	**758**	**808**
Linares	**270**	1684	2376	1881	**332**	**894**	**98**	**201**	1487	**112**	1110	**765**	**116**	**105**	**587**	2096	**776**	**597**	**344**		**384**	**215**	**534**	**624**	**710**	**482**	**725**	**709**	**457**	2816	**221**	**323**	**393**	**167**	**306**	**54**	**209**	**377**	1867	**542**	**412**	**969**	**411**	**456**	**421**	**471**
Los Andes	**646**	1337	2030	1534	**178**	**1270**	**431**	**577**	1863	**436**	**764**	**418**	**271**	**481**	**241**	1750	**430**	**973**	**720**	**384**		**591**	**910**	1000	**363**	**858**	1102	1085	**110**	3192	**161**	**188**	**20**	**217**	**78**	**334**	**585**	**753**	1521	**918**	**129**	**623**	**114**	**832**	**120**	**125**
Los Ángeles	**60**	1878	2579	2087	**539**	**676**	**228**	**127**	1275	**319**	1317	**971**	**322**	**111**	**794**	2303	**983**	**385**	**216**	**215**	**591**		**322**	**412**	**916**	**270**	**513**	**497**	**663**	2604	**428**	**529**	**600**	**374**	**513**	**261**	**153**	**165**	2074	**330**	**619**	1176	**618**	**244**	**628**	**678**
Los Lagos	**288**	2210	2902	2407	**858**	**361**	**547**	**430**	**954**	**638**	1636	1291	**642**	**430**	1113	2622	1302	**64**	**417**	**534**	**910**	**322**		**91**	1236	**145**	**193**	**176**	**983**	2283	**747**	**849**	**919**	**693**	**832**	**580**	**444**	**158**	2393	**67**	**938**	1495	**937**	**119**	**947**	**997**
Osorno	**378**	2274	2975	2497	**948**	**284**	**637**	**523**	**869**	**728**	1726	1381	**732**	**521**	1204	2712	1393	**44**	**507**	**624**	1000	**412**	**91**		1326	**235**	**107**	**91**	1073	2199	**838**	**939**	1009	**784**	**923**	**670**	**534**	**249**	2483	**110**	1029	1585	1057	**210**	1038	1088

Tabla 8.2: Matriz de Distancia de las Ciudades de Chile (B)

	Angol	Antofagasta	Arica	Calama	Cartagena	Castro	Cauquenes	Concepción	Coyhaique	Constitución	Copiapó	Coquimbo	Curico	Chillán	Illapel	Iquique	La Serena	La Unión	Lebu	Linares	Los Andes	Los Ángeles	Los Lagos	Osorno	Ovalle	Pucón	Puerto Montt	Puerto Varas	La Ligua	Punta Arenas	Rancagua	San Antonio	San Felipe	San Fernando	Santiago	Talca	Talcahuano	Temuco	Tocopilla	Valdivia	Valparaíso	Vallenar	Villa Alemana	Villarica	Viña del Mar	Zapallar
Ovalle	972	992	1685	1189	417	1596	756	903	2189	762	419	**94**	597	807	184	1405	**85**	1298	1046	710	363	916	1236	1326		1184	1427	1411	**260**	3518	**486**	**423**	**348**	**542**	**403**	**660**	**910**	1078	1176	1244	**354**	**278**	**339**	1158	**345**	**277**
Pucón	237	2158	2850	2355	**806**	**506**	**495**	**378**	1098	**587**	1584	1239	**590**	**379**	1062	2570	1251	**208**	**365**	**482**	**858**	**270**	**145**	**235**	1184		**337**	**321**	**931**	2428	**696**	**797**	**867**	**642**	**781**	**528**	**392**	**107**	2341	**153**	**887**	1443	**885**	**26**	**896**	**946**
Puerto Montt	**480**	2377	3078	2598	**1050**	**175**	**739**	**626**	**764**	**830**	**1828**	**1482**	**833**	**622**	**1305**	**2814**	**1494**	**145**	**609**	**725**	**1102**	**513**	**193**	**107**	**1427**	**337**		**21**	**1174**	**2300**	**939**	**1040**	**1110**	**885**	**1024**	**771**	**636**	**350**	**2585**	**212**	**1130**	**1686**	**1128**	**311**	**1139**	**1189**
Puerto Varas	**464**	**2385**	**3078**	2 582	**1033**	**190**	**722**	**606**	**782**	**814**	**1812**	**1466**	**817**	**606**	**1289**	**2798**	**1487**	**129**	**593**	**709**	**1085**	**497**	**176**	**91**	**1411**	**321**	**21**		**1158**	**2284**	**923**	**1024**	**1094**	**869**	**1008**	**755**	**620**	**334**	**2568**	**196**	**1114**	**1670**	**1112**	**295**	**1123**	**1173**
La Ligua	**719**	**1234**	**1926**	**1431**	**165**	**1343**	**503**	**650**	**1936**	**509**	**660**	**315**	**344**	**554**	**137**	**1646**	**326**	**1045**	**793**	**457**	**110**	**663**	**983**	**1073**	**260**	**931**	**1174**	**1158**		**3265**	**233**	**170**	**94**	**289**	**150**	**407**	**657**	**825**	**1417**	**991**	**101**	**519**	**86**	**905**	**92**	**34**
Punta Arenas	**2571**	**4451**	**5152**	**4689**	**3140**	**1994**	**2829**	**2700**	**1434**	**2921**	**3919**	**3573**	**2924**	**2713**	**3396**	**4905**	**3585**	**2236**	**2700**	**2816**	**3192**	**2604**	**2283**	**2199**	**3518**	**2428**	**2300**	**2284**	**3265**		**3030**	**3131**	**3201**	**2976**	**3115**	**2862**	**2727**	**2441**	**4675**	**2303**	**3221**	**3777**	**3219**	**2402**	**3230**	**3280**
Rancagua	**483**	**1448**	**2149**	**1657**	**136**	**1104**	**268**	**432**	**1701**	**273**	**887**	**514**	**109**	**318**	**364**	**1873**	**553**	**810**	**558**	**221**	**161**	**428**	**747**	**838**	**486**	**696**	**939**	**923**	**233**	**3030**		**136**	**169**	**54**	**83**	**172**	**422**	**590**	**1644**	**756**	**189**	**746**	**188**	**670**	**198**	**248**
San Antonio	**585**	**1397**	**2030**	**1594**	**9**	**1209**	**369**	**516**	**1802**	**375**	**824**	**478**	**210**	**420**	**301**	**1809**	**490**	**911**	**659**	**323**	**188**	**529**	**849**	**939**	**423**	**797**	**1040**	**1024**	**170**	**3131**	**136**		**197**	**155**	**112**	**273**	**523**	**691**	**1580**	**857**	**85**	**682**	**83**	**771**	**94**	**154**
San Felipe	**655**	**1321**	**2014**	**1518**	**177**	**1279**	**439**	**586**	**1872**	**445**	**748**	**402**	**280**	**490**	**225**	**1734**	**414**	**981**	**729**	**393**	**20**	**600**	**919**	**1009**	**347**	**867**	**1110**	**1094**	**94**	**3201**	**169**	**197**		**225**	**87**	**343**	**593**	**761**	**1505**	**927**	**113**	**607**	**98**	**841**	**104**	**109**
San Fernando	**429**	**1516**	**2209**	**1713**	**165**	**1054**	**214**	**361**	**1646**	**219**	**943**	**597**	**55**	**264**	**420**	**1929**	**609**	**756**	**504**	**167**	**217**	**374**	**693**	**784**	**542**	**642**	**885**	**869**	**289**	**2976**	**54**	**155**	**225**		**139**	**117**	**368**	**536**	**1700**	**702**	**245**	**801**	**243**	**616**	**254**	**304**
Santiago	**568**	**1377**	**2070**	**1574**	**112**	**1191**	**353**	**519**	**1786**	**358**	**803**	**458**	**194**	**403**	**280**	**1798**	**469**	**895**	**643**	**306**	**78**	**513**	**832**	**923**	**403**	**781**	**1024**	**1008**	**150**	**3115**	**83**	**112**	**87**	**139**		**257**	**507**	**675**	**1560**	**841**	**116**	**662**	**107**	**755**	**125**	**164**
Talca	**316**	**1618**	**2319**	**1831**	**282**	**934**	**100**	**262**	**1533**	**106**	**1060**	**715**	**66**	**151**	**538**	**2046**	**726**	**643**	**390**	**54**	**334**	**261**	**580**	**670**	**660**	**528**	**771**	**755**	**407**	**2862**	**172**	**273**	**343**	**117**	**257**		**255**	**423**	**1817**	**588**	**363**	**919**	**361**	**502**	**372**	**422**
Talcahuano	**161**	**1884**	**2577**	**2081**	**533**	**805**	**147**	**16**	**1397**	**253**	**1311**	**965**	**316**	**105**	**788**	**2297**	**977**	**507**	**155**	**209**	**585**	**153**	**444**	**534**	**910**	**392**	**636**	**620**	**657**	**2727**	**422**	**523**	**593**	**368**	**507**	**255**		**287**	**2068**	**452**	**613**	**1170**	**611**	**367**	**622**	**672**
Temuco	**131**	**2038**	**2739**	**2249**	**701**	**514**	**390**	**267**	**1112**	**481**	**1478**	**1133**	**484**	**273**	**956**	**2465**	**1145**	**221**	**260**	**377**	**753**	**165**	**158**	**249**	**1078**	**107**	**350**	**334**	**825**	**2441**	**590**	**691**	**761**	**536**	**675**	**423**	**287**		**2236**	**167**	**781**	**1338**	**779**	**81**	**790**	**840**
Tocopilla	**2129**	**185**	**536**	**156**	**1575**	**2753**	**1913**	**2060**	**3346**	**1919**	**757**	**1102**	**1754**	**1964**	**1367**	**229**	**1092**	**2456**	**2203**	**1867**	**1521**	**2074**	**2393**	**2483**	**1176**	**2341**	**2585**	**2568**	**1417**	**4675**	**1644**	**1580**	**1505**	**1700**	**1560**	**1817**	**2068**	**2236**		**2401**	**1511**	**905**	**1496**	**2315**	**1502**	**1434**
Valdivia	**297**	**2202**	**2903**	**2415**	**866**	**385**	**555**	**452**	**973**	**647**	**1644**	**1289**	**650**	**439**	**1122**	**2630**	**1311**	**66**	**425**	**542**	**918**	**330**	**67**	**110**	**1244**	**153**	**212**	**196**	**991**	**2303**	**756**	**857**	**927**	**702**	**841**	**588**	**452**	**167**	**2401**		**947**	**1503**	**945**	**128**	**956**	**1006**
Valparaíso	**674**	**1319**	**2020**	**1525**	**79**	**1310**	**459**	**639**	**1891**	**464**	**754**	**409**	**300**	**509**	**232**	**1740**	**421**	**1001**	**749**	**412**	**129**	**619**	**938**	**1029**	**354**	**887**	**1130**	**1114**	**101**	**3221**	**189**	**85**	**113**	**245**	**116**	**363**	**613**	**781**	**1511**	**947**		**613**	**25**	**861**	**8**	**72**
Vallenar	**1231**	**722**	**1414**	**919**	**677**	**1855**	**1015**	**1162**	**2448**	**1021**	**148**	**204**	**856**	**1066**	**469**	**1134**	**194**	**1558**	**1305**	**969**	**623**	**1176**	**1495**	**1585**	**278**	**1443**	**1686**	**1670**	**519**	**3777**	**746**	**682**	**607**	**801**	**662**	**919**	**1170**	**1338**	**905**	**1503**	**613**		**598**	**1417**	**604**	**536**
Villa Alemana	**673**	**1313**	**2006**	**1510**	**78**	**1297**	**457**	**604**	**1890**	**463**	**740**	**394**	**298**	**508**	**217**	**1726**	**406**	**999**	**747**	**411**	**114**	**618**	**937**	**1057**	**339**	**885**	**1128**	**1112**	**86**	**3219**	**188**	**83**	**98**	**243**	**107**	**361**	**611**	**779**	**1496**	**945**	**25**	**598**		**859**	**20**	**80**
Villarrica	**211**	**2132**	**2825**	**2329**	**780**	**480**	**469**	**353**	**1073**	**561**	**1558**	**1213**	**564**	**353**	**1036**	**2545**	**1225**	**182**	**340**	**456**	**832**	**244**	**119**	**210**	**1158**	**26**	**311**	**295**	**905**	**2402**	**670**	**771**	**841**	**616**	**755**	**502**	**367**	**81**	**2315**	**128**	**861**	**1417**	**859**		**870**	**920**
Viña del Mar	**683**	**1319**	**2011**	**1516**	**88**	**1308**	**468**	**615**	**1900**	**473**	**745**	**400**	**309**	**518**	**223**	**1731**	**412**	**1010**	**758**	**421**	**120**	**628**	**947**	**1038**	**345**	**896**	**1139**	**1123**	**92**	**3230**	**198**	**94**	**104**	**254**	**125**	**372**	**622**	**790**	**1502**	**956**	**8**	**604**	**20**	**870**		**63**
Zapallar	**733**	**1250**	**1943**	**1448**	**148**	**1358**	**518**	**665**	**1950**	**523**	**677**	**332**	**359**	**568**	**154**	**1663**	**343**	**1060**	**808**	**471**	**125**	**678**	**997**	**1088**	**277**	**946**	**1189**	**1173**	**34**	**3280**	**248**	**154**	**109**	**304**	**164**	**422**	**672**	**840**	**1434**	**1006**	**72**	**536**	**80**	**920**	**63**	

Tabla 8.2 se muestran las Matriz de Distancia de Chile utilizadas para resolver el problema del TSP con CSOP en el presente trabajo, en esta Matriz se muestra las distancias nacionales (distancias reales aproximadas en kilómetros), las cuales fueron obtenidas desde [42].

Tabla 8.3: Matriz de Distancia de las Ciudades de Brasil

	Brasilia	Aracaju	Belém	Belo Horizonte	Boa Vista	Campinas	Campo Grande	Caxias do Sul	Cuiabá	Curitiba	Feira de Santana	Florianópolis	Fortaleza	Goiânia	João Pessoa	Joinville	Juiz de Fora	Londrina	Maceió	Manaus	Natal	Pelotas	Porto Alegre	Porto Velho	Recife	Ribeirão Preto	Rio Branco	Rio de Janeiro	Salvador	Santos	São José dos Campos	São Luís	São Paulo	Teresina	Uberlândia	Vitória
Brasilia		1652	2120	716	4275	921	1134	1900	1133	1366	1415	1673	2378	209	2245	1503	992	1083	1928	3490	2422	2290	2027	2589	2220	706	3123	1148	1446	1087	1087	2157	1015	1789	435	1238
Aracaju	1652		2079	1578	6000	2182	541	3169	2773	2595	322	2892	1183	1849	611	2722	1718	2584	294	5215	788	3559	3296	4229	501	2106	4763	1855	356	2249	2086	1578	2188	1142	2137	1408
Belém	2120	2079		2824	6083	2842	2942	3727	2941	3193	1984	3500	1611	2017	2161	3330	3100	2891	2173	5298	2108	4117	3854	4397	2074	2622	4931	3250	2100	3005	3008	806	2933	947	2367	3108
Belo Horizonte	716	1578	2824		4736	601	1453	1585	1594	1004	1256	1301	2528	906	2171	1131	272	1003	1854	3951	2348	1975	1712	3050	2061	523	3584	434	1372	658	611	2738	586	2302	556	524
Boa Vista	4275	6000	6083	4736		4665	3836	5355	3142	4821	5378	5128	6548	4076	6539	4958	4978	4445	6276	785	6770	5611	5348	1686	6483	4445	2230	5159	5749	4828	4833	6120	4756	6052	4190	5261
Campinas	921	2182	2842	601	4665		1012	1057	1523	476	1866	773	3133	835	2775	600	541	526	2458	3880	2952	1440	1177	2979	2665	238	3513	511	1982	171	174	2879	99	2698	497	959
Campo Grande	1134	541	2942	1453	3836	1012		1525	694	991	2537	1298	3407	935	3357	1128	1512	615	3040	3051	3537	1781	1518	2150	3247	930	2684	1444	2568	1086	1107	2979	1014	2911	894	1892
Caxias do Sul	1900	3169	3727	1585	5355	1057	1525		2213	584	2847	478	4115	1720	3762	606	1498	913	3445	4570	3939	394	150	3669	3652	1215	4203	1426	2963	1054	1089	3764	982	3677	1476	1874
Cuiabá	1133	2773	2941	1594	3142	1523	694	2213		1679	2539	1986	3406	934	3366	1816	1836	1303	3049	2357	3543	2469	2206	1456	3256	1303	1990	2017	2567	1686	1689	2978	1614	2910	1048	2119
Curitiba	1366	2595	3193	1004	4821	476	991	584	1679		2269	300	3541	1186	3188	130	914	379	2871	4036	3365	974	711	3135	3078	681	3669	852	2385	480	515	3230	408	3143	942	1300
Feira de Santana	1415	322	1984	1256	5378	1866	2537	2847	2539	2269		2566	1273	1528	915	2396	1396	2262	598	4893	1092	3237	2974	3907	805	1784	4441	1533	116	1918	1764	1483	1846	1047	1816	1124
Florianópolis	1673	2892	3500	1301	5128	773	1298	478	1986	300	2566		3838	1493	3485	180	1221	686	3168	4343	3662	739	476	3442	3375	988	3976	1144	2682	777	807	3537	705	3450	1249	1597
Fortaleza	2378	1183	1611	2528	6548	3133	3407	4115	3406	3541	1273	3838		2482	688	3668	2668	3356	1075	5763	537	4505	4242	4865	800	2978	5396	2805	1389	3199	3036	1070	3127	634	2707	2397
Goiânia	209	1849	2017	906	4076	835	935	1720	934	1186	1528	1493	2482		2442	1323	1174	884	2105	3291	2619	2110	1847	2390	2332	615	2924	1338	1643	998	1005	2054	926	1986	360	1428
João Pessoa	2245	611	2161	2171	6539	2775	3357	3762	3366	3188	915	3485	688	2442		3315	2311	3177	395	5808	185	4152	3710	4907	120	2699	5356	2448	949	2842	2679	1660	2770	1224	2730	2001
Joinville	1503	2722	3330	1131	4958	600	1128	606	1816	130	2396	180	3668	1323	3315		1046	516	2998	4173	3492	903	640	3272	3205	818	3806	974	2512	607	2679	3367	535	3280	1079	1427
Juiz de Fora	992	1718	3100	272	4978	541	1512	1498	1836	914	1396	1221	2668	1174	2311	1046		1024	1994	4193	2488	1888	1625	3292	2201	676	3826	184	1512	578	415	2874	506	2438	798	519
Londrina	1083	2584	2891	1003	4445	526	615	913	1303	379	2262	686	3356	884	3177	516	1024		2860	3660	3354	1303	1040	1958	3067	478	3293	953	2374	603	621	2928	538	2860	716	1406
Maceió	1928	294	2173	1854	6276	2458	3040	3445	3049	2871	598	3168	1075	2105	395	2998	1994	2860		5491	572	3835	3572	4505	285	2382	5124	2131	632	2525	2362	1672	2453	1236	2413	1684
Manaus	3490	5215	5298	3951	785	3880	3051	4570	2357	4036	4893	4343	5763	3291	5808	4173	4193	3660	5491		5985	4826	4563	901	5698	3660	1445	4374	5009	4043	4046	5335	3971	5267	3405	4476
Natal	2422	788	2108	2348	6770	2952	3537	3939	3543	3365	1092	3662	537	2619	185	3492	2488	3354	572	5985		4329	4066	4999	297	2876	5533	2625	1126	3019	2859	1607	2947	1171	2907	2178
Pelotas	2290	3559	4117	1975	5611	1440	1781	394	2469	974	3237	739	4505	2110	4152	903	1888	1303	3835	4826	4329		271	3925	4042	1605	4459	1816	3353	1444	1479	4154	1372	4067	1866	2264
Porto Alegre	2027	3296	3854	1712	5348	1177	1518	150	2206	711	2974	476	4242	1847	3710	640	1625	1040	3572	4563	4066	271		3662	3779	1342	4196	1553	3127	1181	1216	3891	1109	3804	1603	2001
Porto Velho	2589	4229	4397	3050	1686	2979	2150	3669	1456	3135	3907	3442	4865	2390	4907	3272	3292	1958	4505	901	4999	3925	3662		4712	2759	544	3473	4023	3142	3145	4434	3070	4366	2504	3575
Recife	2220	501	2074	2061	6483	2665	3247	3652	3256	3078	805	3375	800	2332	120	3205	2201	3067	285	5698	297	4042	3779	4712		2589	5243	2338	839	2732	2569	1573	2660	1137	2620	1891
Ribeirão Preto	706	2106	2622	523	4445	238	930	1215	1303	681	1784	988	2978	615	2699	818	676	478	2382	3660	2876	1605	1342	2759	2589		3293	725	1900	391	408	2659	319	2483	281	1048
Rio Branco	3123	4763	4931	3584	2230	3513	2684	4203	1990	3669	4441	3976	5396	2924	5356	3806	3826	3293	5124	1445	5533	4459	4196	544	5243	3293		4007	4457	3676	3679	4968	3604	4900	3038	4109
Rio de Janeiro	1148	1855	3250	434	5159	511	1444	1426	2017	852	1533	1144	2805	1338	2448	974	184	953	2131	4374	2625	1816	1553	3473	2338	725	4007		1649	501	343	3015	429	2579	979	521
Salvador	1446	356	2100	1372	5749	1982	2568	2963	2567	2385	116	2682	1389	1643	949	2512	1512	2374	632	5009	1126	3353	3127	4023	839	1900	4457	1649		2034	1880	1599	1962	1163	1932	1202
Santos	1087	2249	3005	658	4828	171	1086	1054	1686	480	1918	777	3199	998	2842	607	578	603	2525	4043	3019	1444	1181	3142	2732	391	3676	501	2034		169	3042	72	2864	662	954
São José dos Campos	1087	2086	3008	611	4833	174	1107	1089	1689	515	1764	807	3036	1005	2679	2679	415	621	2362	4046	2859	1479	1216	3145	2569	408	3679	343	1880	169		3049	97	2810	669	791
São Luís	2157	1578	806	2738	6120	2879	2979	3764	2978	3230	1483	3537	1070	2054	1660	3367	2874	2928	1672	5335	1607	4154	3891	4434	1573	2659	4968	3015	1599	3042	3049		2970	446	2404	2607
São Paulo	1015	2188	2933	586	4756	99	1014	982	1614	408	1846	705	3127	926	2770	535	506	538	2453	3971	2947	1372	1109	3070	2660	319	3604	429	1962	72	97	2970		2792	590	882
Teresina	1789	1142	947	2302	6052	2698	2911	3677	2910	3143	1047	3450	634	1986	1224	3280	2438	2860	1236	5267	1171	4067	3804	4366	1137	2483	4900	2579	1163	2864	2810	446	2792		2212	2171
Uberlândia	435	2137	2367	556	4190	497	894	1476	1048	942	1816	1249	2707	360	2730	1079	798	716	2413	3405	2907	1866	1603	2504	2620	281	3038	979	1932	662	669	2404	590	2212		1081
Vitória	1238	1408	3108	524	5261	959	1892	1874	2119	1300	1124	1597	2397	1428	2001	1427	519	1406	1684	4476	2178	2264	2001	3575	1891	1048	4109	521	1202	954	791	2607	882	2171	1081	

Las distancias de la Tabla 8.3: Matriz de Distancia de las Ciudades de Brasil fueron obtenidas de [37].

Tabla 8.4: Matriz de Distancia de las Ciudades de USA (A)

	Washington, DC	Albany, New York	Atlanta, Georgia	Albuquerque, New Mexico	Augusta, Maine	Baltimore, Maryland	Billings, Montana	Birmingham, Alabama	Boise, Idaho	Boston, Massachusetts	Buffalo, New York	Charleston, South Carolina	Cheyenne, Wyoming	Chicago, Illinois	Cheveland, Ohio	Columbia, South Calorina	Columbus, Ohio	Dallas/Ft Worth, Texas	Denver, Colorado	Des Moines, Iowa	Detroit, Michigan	El Paso, Texas	Fargo, North Dakota	Grand Junction, Colorado	Hartford, Connecticut	Houston, Texas	Indianapolis, Indiana	Jackson, Mississippi	Jacksonville, Florida	Kansas City, Kansas/Missouri	Las Vegas, Nevada	Little Rock, Arkansas	Los Ángeles, California	Louisville, Kentucky	Memphis, Tennessee	Miami, Florida	Milwaukke, Wisconsin	Mineapolis, Minnesota	Nashville, Tennessee	New Orleans, Louisiana	New York City, New York	Norfolk, Virginia	Oklahoma City, Oklahoma	Omaha, Nebraska	Orlando, Florida	Philadelphia, Pennsylvania	Phoenix, Arizona	Pittsburgh, Pennsylvania	Reno, Nevada	San Antonio, Texas	San Diego, California	San Francisco, California
Washington, DC		645	1000	2984	1161	64	3403	1193	3822	725	661	854	2645	1145	580	774	693	2113	2613	1709	838	3113	2145	3032	580	2209	935	1564	1161	1677	3903	709	4274	967	1371	1709	1306	1758	1064	1774	387	306	2145	1838	1371	209	3709	403	4145	2564	4193	4580
Albany, New York	645		1629	3290	709	548	3387	1725	4064	274	483	1419	2887	1322	774	1322	1032	2709	2951	1903	1048	3580	2403	3387	177	2655	1306	2177	1725	2096	4242	2193	4597	1387	1983	2258	1500	2016	1661	2322	258	822	2419	2113	1983	419	4016	725	4451	3177	4597	4806
Atlanta, Georgia	1000	1629		2258	2145	1048	2903	241	3580	1790	1467	467	2387	1145	1177	338	951	1322	2306	1548	1177	2322	2242	2677	1548	1274	854	645	500	1322	3193	887	3532	677	612	1064	1290	1806	403	774	1371	903	1338	1629	693	1209	2951	1096	3887	1613	3467	4000
Albuquerque, New Mexico	2984	3290	2258		3967	3048	1596	2016	1516	3580	2855	2742	871	2113	2564	2613	2354	1032	709	1564	2516	435	2080	693	3387	1371	2048	1693	2645	1258	851	1419	1306	2080	1629	3177	2225	1967	1983	1935	3226	3048	871	1435	2806	3145	741	2629	1645	1177	1306	1790
Augusta, Maine	1161	709	2145	3967		1080	3984	2435	4774	435	1177	1967	3580	2016	1403	1951	1629	3258	3613	2532	1709	4242	3048	4048	596	3419	1967	2822	2371	2774	4887	2855	5322	2016	2629	2951	2129	2645	2290	2919	822	1435	3113	3758	2613	951	4709	1387	5064	3693	5274	5451
Baltimore, Maryland	64	548	1048	3048	1080		3096	1242	3887	693	596	919	2693	1161	580	838	693	2193	2645	1677	822	3193	2177	3064	516	2258	919	1629	1209	1725	3871	1677	4274	1032	1467	1758	1274	1790	1129	1838	322	370	3113	1854	1435	161	3726	403	4226	2629	4322	4548
Billings, Montana	3403	3387	2903	1596	3984	3096		2871	983	3548	2774	3516	741	1983	2596	3322	2596	2290	887	1580	2451	2032	983	1080	3387	2564	2274	2774	3548	1677	1693	2290	2000	2467	2345	4064	1871	1338	2580	2951	3290	3355	1854	1467	3596	3193	1935	2677	1548	2451	2113	1919
Birmingham, Alabama	1193	1725	241	2016	2435	1242	2871		3338	1983	1451	709	2290	1064	1177	580	919	1032	2145	1338	1225	2048	2096	2629	1742	1129	774	403	741	1177	2951	693	3306	580	419	1209	1193	1725	306	564	1580	1145	1161	1419	887	1403	2709	1274	3677	1387	3209	3822
Boise, Idaho	3822	4064	3580	1516	4774	3887	983	3338		4338	3564	4016	1177	2758	3274	3855	3177	2564	1354	2193	3193	1935	1951	1016	4209	2935	2984	3290	3984	2290	1080	2855	1371	3113	2935	4629	2806	2387	3193	3467	4016	4064	2322	2000	4258	3919	1580	3435	693	2693	1580	1048
Boston, Massachusetts	725	274	1790	3580	435	693	3548	1983	4338		758	1516	3096	1613	1064	1548	1290	2822	3226	2161	1290	3838	2661	3661	177	2951	1500	2354	1871	2322	4435	2322	4871	1548	2161	2451	1758	2242	1758	2435	338	935	2725	2371	2096	516	4306	919	4629	3258	4645	5048
Buffalo, New York	661	483	1467	2855	1177	596	2774	1451	3564	758		1500	2419	871	306	1290	532	2193	2500	1371	580	3177	1919	2887	677	2403	822	1806	1758	1629	3677	1709	4161	887	1483	2258	1016	1532	1161	2032	596	967	1967	1645	1935	580	3580	354	3951	2629	4080	4306
Charleston, South Carolina	854	1419	467	2742	1967	919	3516	709	4016	1516	1500		2758	1467	1177	177	1048	1758	2774	1935	1354	2742	2435	3161	1338	1693	1161	1080	387	1790	3677	1338	4016	938	1129	951	1645	2145	871	1177	1242	709	1838	2048	612	1064	3419	1048	4387	2064	3903	4548
Cheyenne, Wyoming	2645	2887	2387	871	3580	2693	741	2290	1177	3096	2419	2758		1580	2145	2677	2096	1419	161	1032	2000	1274	1258	564	2984	1790	1742	2048	2822	1096	1371	1725	1806	1935	1838	3467	1613	1419	2032	2225	2822	2903	1112	806	3016	2838	1516	2306	1564	1709	1919	1919
Chicago, Illinois	1145	1322	1145	2113	2016	1161	1983	1064	2758	1613	871	1467	1580		564	1290	580	1483	1645	548	451	2354	1048	2048	1467	1758	306	1209	1629	871	2871	1064	3306	483	871	2258	145	661	758	1483	1306	1403	1338	774	1854	1274	2806	774	3145	1951	3371	3500
Cheveland, Ohio	580	774	1177	2564	1403	580	2596	1177	3274	1064	306	1177	2145	564		983	225	1919	2193	1064	274	2806	1613	2596	903	2113	516	1516	1467	1322	3371	1403	3838	564	1177	2016	709	1225	854	1709	758	822	1661	1338	1693	693	3274	209	3645	2338	3855	4000
Columbia, South Calorina	774	1322	338	2613	1951	838	3322	580	3855	1548	1290	177	2677	1290	983		854	1661	2613	1790	1161	2629	2306	2967	1306	1661	983	967	483	1645	3532	1225	3903	806	1000	1032	1419	1951	709	1112	1161	629	1709	1903	709	983	3274	951	4209	1983	3838	4468
Columbus, Ohio	693	1032	951	2354	1629	693	2596	919	3177	1290	532	1048	2096	580	225	854		1693	2000	1064	306	2580	1532	2403	1064	1887	290	1290	1338	1096	3258	1177	3613	338	951	1887	758	1258	612	1483	903	903	1483	1274	1548	774	3064	306	3564	2113	3645	3919
Dallas/Ft Worth, Texas	2113	2709	1322	1032	3258	2193	2290	1032	2564	2822	2193	1758	1419	1483	1919	1661	1693		1258	1129	1871	1000	1758	1580	2725	403	1435	661	1677	822	1983	516	2258	1338	725	2161	1645	1532	1064	838	2516	2177	338	1064	1774	2322	1613	1951	2693	435	2177	2822
Denver, Colorado	2613	2951	2306	709	3613	2645	887	2145	1354	3226	2500	2774	161	1645	2193	2613	2000	1258		1080	2064	1112	1419	403	3226	1661	1709	1935	2806	983	1225	1516	1661	1806	1677	3403	1677	1483	1903	2064	2887	2855	1016	871	3032	2806	1306	2306	1661	1532	1774	2032
Des Moines, Iowa	1709	1903	1548	1564	2532	1677	1580	1338	2193	2161	1371	1935	1032	548	1064	1790	1064	1129	1080		951	1822	774	1483	2016	1516	774	1338	2000	322	2306	903	2758	951	1000	2564	596	403	1112	1580	1806	1854	887	209	2193	1758	2306	1258	2580	1580	2855	2951
Detroit, Michigan	838	1048	1177	2516	1709	822	2451	1225	3193	1290	580	1354	2000	451	274	1161	306	1871	2064	951		2693	1483	2451	1177	2064	451	1548	1693	1242	3258	1419	3693	612	1161	2242	580	1112	871	1725	1048	1177	1661	1177	1887	983	3242	483	3548	2338	3822	3871
El Paso, Texas	3113	3580	2322	435	4242	3193	2032	2048	1935	3838	3177	2742	1274	2354	2806	2629	2580	1000	1112	1822	2693		2338	1080	3693	1193	2290	1661	2548	1516	1161	1516	1322	2338	1725	3064	2484	2354	2064	1774	3467	3145	1096	2016	2709	3338	709	2871	1887	903	1177	1935
Fargo, North Dakota	2145	2403	2242	2080	3048	2177	983	2096	1951	2661	1919	2435	1258	1048	1613	2306	1532	1758	1419	774	1483	2338		1790	2516	2096	1338	2113	2725	1000	2484	1677	2919	1516	1822	3226	919	387	1790	2225	2338	2371	1403	693	2919	2306	2725	1822	2548	2145	3016	2935
Grand Junction, Colorado	3032	3387	2677	693	4048	3064	1080	2629	1016	3661	2887	3161	564	2048	2596	2967	2403	1580	403	1483	2451	1000	1790		3467	1967	2145	2274	3161	1387	822	1854	1258	2209	2048	3709	2080	1871	2290	2371	3290	3258	1322	1274	3338	3209	935	2725	1209	1758	1371	1580
Hartford, Connecticut	580	177	1548	3387	596	516	3387	1742	4209	177	677	1338	2984	1467	903	1306	1064	2725	3226	2016	1177	3693	2516	3467		2790	1354	2177	1693	2161	4290	2161	4677	1403	1935	2290	1613	2129	1613	2306	193	790	2645	2209	1774	338	4145	758	4516	3080	4677	4968
Houston, Texas	2209	2855	1274	1371	3419	2258	2564	1129	2935	2951	2403	1693	1790	1758	2113	1661	1887	403	1661	1516	2064	1193	2096	1967	2790		1613	677	1435	1193	2371	693	2484	1516	919	1919	1903	1903	1258	564	2596	2177	741	1387	1580	2435	1871	2209	3032	322	2403	3080
Indianapolis, Indiana	935	1306	854	2048	1967	919	2274	774	2984	1500	822	1161	1742	306	516	983	290	1435	1709	774	451	2290	1338	2145	1354	1613		1080	1354	806	2967	983	3322	177	758	1919	435	951	451	1290	1177	1145	1193	983	1564	1064	2790	580	3322	1919	3355	3693
Jackson, Mississippi	1564	2177	645	1693	2822	1629	2774	403	3290	2354	1806	1000	2048	1209	1516	967	1290	661	1935	1338	1548	1661	2113	2274	2177	677	1080		983	1112	2629	419	2935	951	338	1467	1338	1709	661	338	1967	1532	903	1403	1129	1790	2354	1516	3355	1080	2838	3467
Jacksonville, Florida	1161	1725	500	2645	2371	1209	3548	741	3984	1871	1758	387	2822	1629	1467	483	1338	1677	2806	2000	1693	2548	2725	3161	1693	1435	1354	983		1838	3613	1322	3790	1177	1112	564	1790	2354	903	903	1532	1000	1854	2129	225	1371	3226	1338	1338	1742	3806	4435
Kansas City, Kansas/Missouri	1677	2096	1322	1258	2774	1725	1677	1177	2290	2322	1629	1790	1096	871	1322	1645	1096	822	983	322	1242	1516	1000	1387	2161	1193	806	1112	1838		2209	677	2548	822	774	2387	903	709	935	1354	1983	1871	564	306	2016	1887	2000	1403	2645	1258	2564	3000
Las Vegas, Nevada	3903	4242	3193	851	4887	3871	1693	2951	1080	4435	3677	3677	1371	2871	3371	3532	3258	1983	1225	2306	3258	1161	2484	822	4290	2371	2967	2629	3613	2209		2354	435	3016	2580	4145	2903	2677	2919	2790	4145	4000	1790	2096	3790	4000	467	3564	725	2080	548	919
Little Rock, Arkansas	709	2193	887	1419	2855	1677	2290	693	2855	2322	1709	1338	1725	1064	1403	1225	1177	516	1516	903	1419	1516	1677	1854	2161	693	983	419	1322	677	2354		2725	822	225	1871	1242	1338	564	677	2016	1645	548	935	1548	1838	2145	1451	645	2693	3209	3629
Los Ángeles, California	4274	4597	3532	1306	5322	4274	2000	3306	1371	4871	4161	4016	1806	3306	3838	3903	3613	2258	1661	2758	3693	1322	2919	1258	4677	2484	3322	2935	3790	2548	435	2725		3516	2919	4387	3338	3000	3242	3000	4500	4338	2161	2532	3919	4355	629	3919	758	2242	209	629
Louisville, Kentucky	967	1387	677	2080	2016	1032	2467	580	3113	1548	887	938	1935	483	564	806	338	1338	1806	951	612	2338	1516	2209	1403	1516	177	951	1177	822	3016	822	3516		612	1758	612	1145	274	1129	1242	1048	1209	1129	1387	1145	2822	629	3451	1774	3371	3790
Memphis, Tennessee	1371	1983	612	1629	2629	1467	2345	419	2935	2161	1483	1129	1838	871	1177	1000	951	725	1677	1000	1161	1725	1822	2048	1935	919	758	338	1112	774	2580	225	2919	612		1613	1000	1467	338	661	1774	1419	774	1032	1258	1629	2371	1225	3274	1177	2919	3419
Miami, Florida	1709	2258	1064	3177	2951	1758	4064	1209	4629	2451	2258	951	3467	2258	2016	1032	1887	2161	3403	2564	2242	3064	3226	3709	2290	1919	1919	1467	564	2387	4145	1871	4387	1758	1613		2354	2855	1467	1387	2145	1564	2419	2645	370	1983	3790	1983	4829	2242	4322	4984

Tabla 8.5: Matriz de Distancia de las Ciudades de USA (B)

	Washington, DC	Albany, New York	Atlanta, Georgia	Albuquerque, New Mexico	Augusta, Maine	Baltimore, Maryland	Billings, Montana	Birmingham, Alabama	Boise, Idaho	Boston, Massachusetts	Buffalo, New York	Charleston, South Carolina	Cheyenne, Wyoming	Chicago, Illinois	Cleveland, Ohio	Columbia, South Calorina	Columbus, Ohio	Dallas/Ft Worth, Texas	Denver, Colorado	Des Moines, Iowa	Detroit, Michigan	El Paso, Texas	Fargo, North Dakota	Grand Junction, Colorado	Hartford, Connecticut	Houston, Texas	Indianapolis, Indiana	Jackson, Mississippi	Jacksonville, Florida	Kansas City, Kansas/Missouri	Las Vegas, Nevada	Little Rock, Arkansas	Los Ángeles, California	Louisville, Kentucky	Memphis, Tennessee	Miami, Florida	Milwaukke, Wisconsin	Minneapolis, Minnesota	Nashville, Tennessee	New Orleans, Louisiana	New York City, New York	Norfolk, Virginia	Oklahoma City, Oklahoma	Omaha, Nebraska	Orlando, Florida	Phidadelphia, Pennsylvania	Phoenix, Arizona	Pittsburgh, Pennsylvania	Reno, Nevada	San Antonio, Texas	San Diego, California	San Francisco, California
Milwaukke, Wisconsin	1306	1500	1290	2225	2129	1274	1871	1193	2806	1758	1016	1645	1613	145	709	1419	758	1645	1677	596	580	2484	919	2080	1613	1903	435	1338	1790	903	2903	1242	3338	612	1000	2354		548	887	1661	1435	1532	1419	806	1967	1403	2855	903	3161	2080	3435	3500
Minneapolis, Minnesota	1758	2016	1806	1967	2645	1790	1338	1725	2387	2242	1532	2145	1419	661	1225	1951	1258	1532	1483	403	1112	2354	387	1871	2129	1903	951	1709	2354	709	2677	1338	3000	1145	1467	2855	548		1338	2177	1967	1983	1274	612	2532	1935	2709	1435	2951	2016	3226	3193
Nashville, Tennessee	1064	1661	403	1983	2290	1129	2580	306	3193	1758	1161	871	2032	758	854	709	612	1064	1903	1112	871	2064	1790	2290	1613	1258	451	661	903	935	2919	564	3242	274	338	1467	887	1338		854	1451	1080	1096	1225	1112	1274	2693	919	3580	1500	3226	3758
New Orleans, Louisiana	1774	2322	774	1935	2919	1838	2951	564	3467	2435	2032	1177	2225	1483	1709	1112	1483	838	2064	1580	1725	1774	2225	2371	2306	564	1290	338	903	1354	2790	677	3000	1129	661	1387	1661	2177	854		2161	1677	1112	1661	1048	1983	2419	1838	3596	887	2967	3677
New York City, New York	387	258	1371	3226	822	322	3290	1580	4016	338	596	1242	2822	1306	758	1161	903	2516	2887	1806	1048	3467	2338	3290	193	2596	1177	1967	1532	1983	4145	2016	4500	1242	1774	2145	1435	1967	1451	2161		596	2387	2016	1758	177	3951	612	4371	2935	4516	4726
Norfolk, Virginia	306	822	903	3048	1435	370	3355	1145	4064	935	967	709	2903	1403	822	629	903	2177	2855	1854	1177	3145	2371	3258	790	2177	1145	1532	1000	1871	4000	1645	4338	1048	1419	1564	1532	1983	1080	1677	596		2209	2129	1242	435	3790	677	4500	2500	4322	4839
Oklahoma City, Oklahoma	2145	2419	1338	871	3113	3113	1854	1161	2322	2725	1967	1838	1112	1338	1661	1709	1483	338	1016	887	1661	1096	1403	1322	2645	741	1193	903	1854	564	1790	548	2161	1209	774	2419	1419	1274	1096	1112	2387	2209		741	1983	2242	1580	1790	2516	774	2145	2677
Omaha, Nebraska	1838	2113	1629	1435	3758	1854	1467	1419	2000	2371	1645	2048	806	774	1338	1903	1274	1064	871	209	1177	2016	693	1274	2209	1387	983	1403	2129	306	2096	935	2532	1129	1032	2645	806	612	1225	1661	2016	2129	741		2274	1935	2177	1500	2371	1516	2629	2725
Orlando, Florida	1371	1983	693	2806	2613	1435	3596	887	4258	2096	1935	612	3016	1854	1693	709	1548	1774	3032	2193	1887	2709	2919	3338	1774	1580	1564	1129	225	2016	3790	1548	3919	1387	1258	370	1967	2532	1112	1048	1758	1242	1983	2274		1596	3355	1580	4580	1887	3887	4629
Phidadelphia, Pennsylvania	209	419	1209	3145	951	161	3193	1403	3919	516	580	1064	2838	1274	693	983	774	2322	2806	1758	983	3338	2306	3209	338	2435	1064	1790	1371	1887	4000	1838	4355	1145	1629	1983	1403	1935	1274	1983	177	435	2242	1935	1596		3822	500	4274	2806	4468	4677
Phoenix, Arizona	3709	4016	2951	741	4709	3726	1935	2709	1580	4306	3580	3419	1516	2806	3274	3274	3064	1613	1306	2306	3242	709	2725	935	4145	1871	2790	2354	3226	2000	467	2145	629	2822	2371	3790	2855	2709	2693	2419	3951	3790	1580	2177	3355	3822		3371	1177	1613	564	1225
Pittsburgh, Pennsylvania	403	725	1096	2629	1387	403	2677	1274	3435	919	354	1048	2306	774	209	951	306	1951	2306	1258	483	2871	1822	2725	758	2209	580	1516	1338	1403	3564	1451	3919	629	1225	1903	903	1435	919	1838	612	677	1790	1500	1580	500	3371		3822	2387	3935	4209
Reno, Nevada	4145	4451	3887	1645	5064	4226	1548	3677	693	4629	3951	4387	1564	3145	3645	4209	3564	2693	1661	2580	3548	1887	2548	1209	4516	3032	3322	3355	1338	2645	725	645	758	3451	3274	4829	3161	2951	3580	3596	4371	4500	2516	2371	4580	4274	1177	3822		2790	967	370
San Antonio, Texas	2564	3177	1613	1177	3693	2629	2451	1387	2693	3258	2629	2064	1709	1951	2338	1983	2113	435	1532	1580	2338	903	2145	1758	3080	322	1919	1080	1742	1258	2080	2693	2242	1774	1177	2242	2080	2016	1500	887	2935	2500	774	1516	1887	2806	1613	2387	2790		2096		2806
San Diego, California	4193	4597	3467	1306	5274	4322	2113	3209	1580	4645	4080	3903	1919	3371	3855	3838	3645	2177	1774	2855	3822	1177	3016	1371	4677	2403	3355	2838	3806	2564	548	3209	209	3371	2919	4322	3435	3226	3226	2967	4516	4322	2145	2629	3887	4468	564	3935	967	2096		822
San Francisco, California	4580	4806	4000	1790	5451	4548	1919	3822	1048	5048	4306	4548	1919	3500	4000	4468	3919	2822	2032	2951	3871	1935	2935	1580	4968	3080	3693	3467	4435	3000	919	3629	629	3790	3419	4984	3500	3193	3758	3677	4726	4839	2677	2725	4629	4677	1225	4209	370	2806	822	

Los distancias de la Tabla 8.4: Matriz de Distancia de las Ciudades de USA (A) y la

Tabla 8.5: Matriz de Distancia de las Ciudades de USA (B) fueron obtenidas desde [38].

8.3 Consideraciones Previas a la Simulación Computacional

Para la implementación de los script mostrados en el anexo del presente informe se utiliza la plataforma de desarrollo Mozart, la versión utilizada de la plataforma de desarrollo corresponde a la versión 1.4.0 y las pruebas realizadas se ejecutaron en un computador con procesador AMD Dual Core 2 de 5200 con 2 GB de memoria Ram. Los indicadores de Desempeños aplicados en la presente sección, corresponde a la evaluación del rendimiento del Prototipo de Sistema frente a las diversas Estrategias de Distribución. Con el objetivo de medir y establecer tales diferencias es necesario establecer algún indicador de desempeño que permita reflejar con claridad el efecto de aplicación de una estrategia de enumeración u otra estrategia.

- Tiempo: mide el tiempo requerido para resolver el problema, asignado a 1 minuto dado que es un tiempo razonable de espera.
- Número de Nodos: cuenta la cantidad de nodos o espacios generados para encontrar la solución del problema, incluyendo los nodos que conducen a una solución y las malas que obligan a retroceder.

- Número de Fallos: cuenta la cantidad de fallos producidos antes de encontrar la primera solución o todas las soluciones.

- Profundidad: indica la profundidad el árbol binario generado por Mozart/OZ.

- Mejor Óptimo: indica el mejor optimo encontrado por el algoritmo B&B.

8.4 Resultado de Prueba de Rendimientos al Prototipo de Sistema

Tabla 8.6: Resultados de las Pruebas de Heurísticas (A)

	Distancias de 46 Ciudades de Chile (Ciudad Origen: Villa Alemana)					
	Fallos	**Nodos**	**Óptimos**	**Profundidad**	**Tiempo**	**Mejor Óptimo**
E1	587544	587597	11	73	1 minuto	37745 kilómetros
E2	538647	538703	15	73	1 minuto	36810 kilómetros
E3	591671	591753	40	73	1 minuto	43561 kilómetros
E4	484717	484780	20	73	1 minuto	39295 kilómetros
E5	592294	592345	8	89	1 minuto	37980 kilómetros
E6	624494	624553	16	89	1 minuto	37283 kilómetros
E7	580799	580868	25	89	1 minuto	44744 kilómetros
E8	620536	620594	16	89	1 minuto	39732 kilómetros
E9	607969	608021	11	73	1 minuto	37745 kilómetros
E10	587904	587962	15	73	1 minuto	36810 kilómetros
E11	555565	555647	39	73	1 minuto	43562 kilómetros
E12	538803	538863	20	73	1 minuto	39295 kilómetros

Tabla 8.7: Resultados de las Pruebas de Heurísticas (B)

	Distancias de 36 Ciudades de Brasil (Ciudad Origen: Porto Alegre)					
	Fallos	**Nodos**	**Óptimos**	**Profundidad**	**Tiempo**	**Mejor Óptimo**
E1	759419	759480	37	64	1 minuto	79049 kilómetros
E2	760235	760308	43	64	1 minuto	77743 kilómetros
E3	658063	658110	17	63	1 minuto	66943 kilómetros
E4	615162	615243	48	63	1 minuto	78743 kilómetros
E5	737681	737742	29	79	1 minuto	79932 kilómetros
E6	755023	755086	29	80	1 minuto	81970 kilómetros
E7	749274	749320	14	80	1 minuto	67723kilómetros
E8	773257	773342	53	80	1 minuto	79415 kilómetros
E9	666681	666747	37	63	1 minuto	79049 kilómetros
E10	666067	666141	43	63	1 minuto	77743 kilómetros
E11	800093	800141	17	64	1 minuto	66943 kilómetros
E12	645270	645349	48	63	1 minuto	78743 kilómetros

Tabla 8.8: Resultados de las Pruebas de Heurísticas (C)

	Distancias de 52 Ciudades de USA (Ciudad Origen: New York City – New York)					
	Fallos	**Nodos**	**Óptimos**	**Profundidad**	**Tiempo**	**Mejor Óptimo**
E1	462867	462951	35	79	1 minuto	96362 kilómetros
E2	504475	504569	45	79	1 minuto	91589 kilómetros
E3	627946	628012	18	79	1 minuto	90011 kilómetros
E4	559692	559791	52	79	1 minuto	90895kilómetros
E5	563119	563200	33	95	1 minuto	97022 kilómetros
E6	514187	514275	39	94	1 minuto	91621 kilómetros
E7	536230	536289	11	94	1 minuto	91720 kilómetros
E8	472549	472625	27	94	1 minuto	93830 kilómetros
E9	487224	487307	35	79	1 minuto	96362 kilómetros
E10	655630	655722	45	79	1 minuto	91589 kilómetros
E11	639124	639190	18	79	1 minuto	90011 kilómetros
E12	431360	431459	52	79	1 minuto	90895 kilómetros

En la Tabla 8.7 y la Tabla 8.8 se puede apreciar que las estrategias de distribución E_2 que representa la Estrategia de Distribución compuesta por la Heurística de Selección DMi junto a la Heurística de Selección de Valores MaVal y la E_{10} que representa la Estrategia de Distribución compuesta por la Heurística de Selección BOr junto a la Heurística de Selección de Valores MaVal, son con mejor rendimiento con respecto a las demás estrategias de distribución utilizadas, esto es debido a la naturaleza del problema, mas especifico en la distribución de las ciudades de los países Estados Unidos y Brasil, en contraparte la Tabla 8.6 que representa los resultados obtenidos de las ciudades de Chile, las estrategias con mejor rendimiento fueron la E_3, que representa la Estrategia de Distribución compuesta por la Heurística de Selección BOr junto a la Heurística de Selección de Valores MedVal, y la E_{10}, esto debido a la distribución de las ciudades en Chile en donde es más linear, debido que están distribuidas de norte a sur, y en si con menos alternativa en comparación con las ciudades de Estados Unidos y Brasil donde hay más ciudades con la misma distancia a una ciudad central del país, por lo tanto es menos lineal que la distribución de las ciudades de Chile.

En referente al rendimiento general del prototipo, a pesar que en si Oz consume muchos recurso del procesador de un PC, por lo mismo es recomendado tomar las especificaciones de sistema descritas en el presente informe como requerimiento mínimo ideal para el correcto funcionamiento del prototipo de sistema, se a obtenido un rendimiento razonable, ya que se definido el tiempo de 1 minuto como el máximo comprensible que deberá esperar el usuario para generar la base de dato del sistema, ver en el Anexo del presente informe el manual de usuario adjunto, y dado que en ese tiempo el algoritmo B&B, genera un muestreo aceptables de soluciones optimas y soluciones analizadas, aproximadamente alrededor de 500000 soluciones analizadas, con más tiempo esto aumentaría, pero significaría la espera mayor por parte del usuario. En referente a la interfaz grafica del prototipo de sistema, el cual permite al usuario elegir el país y la ciudad de origen para generar el resultado del TSP, véase el manual de usuario adjuntado en el anexo del presente informe, el tiempo de mostrara los datos de la matriz de distancia del país seleccionado y la creación de la matriz modificada según la ciudad de origen seleccionada y mostrar el resultado del TSP de dicha matriz, es casi óptima ya que se demora unos segundos en generarlos.

Capítulo 9

Conclusiones y Trabajos Futuros

A modo de conclusión, puede decirse que no hay que dejarse engañar por lo pequeño que pueda resultar un enunciado de un problema cualquiera, el TSP esconde una dificultad interesante. Con este trabajo se pretendió conocer más sobre lo realizado previamente por personajes que han hecho importantes aportes a la solución de problemas complejos, desarrollando algoritmos y métodos para atacar problemas como el TSP, y en qué forma se ha ido evolucionando la Programación con Restricciones pasando de CP a CSP y por último a CSOP. Éste último es el que mejor se adapto al TSP, y en éste trabajo se ha dado a conocer estos conceptos a los futuros lectores.

Es importante saber que herramientas disponibles de forma libre para la investigación de problemas de este tipo existen en la red. Herramientas tales como Mozart/Oz, que como lenguaje, permitirá expresar y probar de manera práctica el modelado del problema TSP con CSOP y verificar si vamos por buen camino o no. Es más, uno de los algoritmos nos da una luz de cómo aprovechar las ventajas de utilizar Mozart/Oz y su habilidad de distribuir sub-espacios de cómputo para distribuir la carga de trabajo a la hora de encontrar soluciones: Branch and Bound. Teniendo en cuenta que hasta la fecha no se ha intentado de resolver problema del TSP con este lenguaje, aunque si se ha atacado con el lenguaje Oracle en lo que se respecta a la familia de lenguaje que soporta CP. Oz en sí es un lenguaje gratuito que a lo largo de su vida ha pasado por diferentes organizaciones, encargadas de su elaboración y propagación, aún así es un lenguaje poco conocido y encerrado en el mundo universitario y en el laboratorio computacional, faltando más darse a conocer para llegar a ser utilizado de manera masiva en el campo laborar, siendo su última actualización a mediados del año 2008, aunque en el transcurso de este proyecto se contacto con uno de los creadores de Oz, descubriendo que a lo largo del año 2010 habrá una nueva versión.

También se ha entendido la complejidad que es aprender un nuevo paradigma como es la Programación con Restricciones y sus arias, con problemas clásicos de CSP, nos damos cuenta de la profundidad de este paradigma y las ventajas que brinda a la hora de afrentar dichos problemas en comparación a paradigmas más comunes, como el estructurado y el orientado a objetos, y su capacidad de modelar en forma matemática problemas reales como N-Reinas y Coloración de Mapas, y diversos problemas conocidos de tipo N-Hard por su complejidad computacional. También a lo largo del proyecto se ha podido apreciar las ventajas y desventajas de la programación de restricciones y su aplicación al problema del Vendedor Viajero, pero todo lo expuesto en el presente trabajo sirve como base para futuros proyectos y/o estudios, por lo tanto, aplicando el resultado obtenido en el proyecto en el área de TSP y aplicarle en híbridos entre CSOP y otras

técnicas de inteligencia artificial, como Algoritmos Genéticos, este hibrido fue abarcado de modo de introducción en la sección de conceptos en el presente proyecto, ya que no es el objetivo de este trabajo entrar en detalles y realizar una aplicación de Gas entrar en la área de aplicar algoritmos y heurísticas para mejorar los resultados obtenidos de algoritmo como B&B, estas técnicas fueron explica a lo largo del presente informe. Tomando en cuenta en cuenta todo lo descrito en el presente informe, nos podemos dar cuenta de la facilidad de la adaptación del problema del vendedor viajero modelado en CSOP y adaptado al lenguaje Oz, teniendo este ultimo su complejidad, debido al mal diseño del manual oficial de Mozart-oz, siendo solamente un introducción al lenguaje, escrito para un público de conocimiento elevados en el campo de informática, específicamente inteligencia artificial y problemas de la naturaleza N-hard, aun así, hay que atreverse a entrar a aprender paradigmas nuevos de la programación como son el CP, ya que la facilidad del moldeamiento matemático, usos de hebras y mesclas de paradigmas como programación tradicional y objetos entre otros facilita la adaptación de problemas complejos y reales de la vida cotidiana a su expresión computacional, logrando optimizar los resultados obtenido a lo largo de la historias pro varios científicos.

Como resultados obtenidos se llegó a la conclusión de que el ocupar Programación con Restricciones trae beneficios a la hora de modelar problemas reales y matemáticos en comparación con otros paradigmas computacionales en término de disminución de complejidad de modelamiento del problema y la capacidad de generar una muestra considerable de soluciones posibles, en el caso del prototipo de sistema que obtuvo una muestra de alrededor de 500000 soluciones posibles por ciudad y a luego rescatando las soluciones optimas de ellos.

Para trabajos futuros se puede trabajar en la implementación de heurísticas complementarias que mejoran recorridos y construyen recorridos en los conjuntos de óptimos obtenidos mediante B&B en el presente trabajo, así logrando reducir la distancia del resultado del TSP, esto se podría adaptar a la nueva versión de Mozart/Oz cuyo lanzamiento está planificado para este año. Por otra parte, se puede trabajar en la integración del modulo generador de resultados del TSP y el modulo que genera las matriz de distancias según la selección del país y la ciudad de origen mediante la interfaz del prototipo de sistema, esto debido a la incompatibilidad de la extensión del modulo SearchBest con respecto al modulo QTk encargado de la interfaz grafica que existe en la actualidad, todo esto a espera de la nueva versión de Mozart/Oz y la posibilidad compatibilidad en su nueva versión, tendiendo en consideración que actualmente existen la mayoría de las expansión no son compatible con Windows ni Linux, solo con Mac OS X.

Queda mucho todavía, y es difícil competir contra los resultados obtenidos de años de investigación y recursos, pero ahí estuvo el desafío y una meta de alcanzar: encontrar una manera diferente de resolver TSP y si es posible, mejorar el rendimiento de las soluciones actuales, eso sí siempre se rigió a los requerimientos del cliente de este proyecto: la Empresa Neogística S.A.

Capítulo 10

Referencias

[1] F. Barbe, M. A. Salido, "Problemas de Satisfacción de Restricciones (CSP)", Inteligencia Artificial: Técnicas, métodos y aplicaciones. Ed. McGraw-Hill, paginas: 385-432, 2008.

[2] R. Barták. "Constraint Programming: In Pursuit of the Holy Grail". In Proccedings of Week of Doctoral Students (WDS99), Part IV, MatFyzPress, Prague, June 1998, pages: 555-564.

[3] Miguel and Q. Shen, "Solution Techniques for Constraint Satisfaction Problems: Foundations", School of Artificial Intelligence, Division of Informatics , University of Edinburgh, Edinburgh, pages: 243-267 ISSN:0269-2621, 2001.

[4] F. Barbe, M. A. Salido, "Introduction to constraint programming", Inteligencia Artifical. Revista Iberoamericana de Inteligencia Artificial Vol 7, Numero 020, pages: 13-30, 2003

[5] A. K. Mackworth. "Consistency in network of relations". Artificial Intelligence 8, pages: 99-118, 1977.

[6] E. C. Freuder. "A sufficient condition for backtrack-bounded search", Journal of the ACM (JACM) Vol 29, Department of Computer Science, University of New Hampshire, Durham, New Hampshire, ISSN:0004-5411 pages: 24-32, 1982.

[7] E. C. Freuder. "Synthesizing constraint expressions", Communications of the ACM, Department of Computer Science, University of New Hampshire, Durham, New Hampshire, Vol 21, pages: 958-966, ISSN:0001-0782, 1978.

[8] R. Debruyne and C. Bessière. "Some practicable filtering techniques for the constraint satisfaction problem", In proceedings of the 15th IJCAL, pages: 412-417, 1997.

[9] U. Montanari, "Networks of constraints: fundamental properties and applications to picture processing", Information Sciences, Vol 7, pages: 95-132, 1974.

[10] J. R. Bitner, & E. M. Reingold, "Backtrack Programming Techniques", Communications of the ACM, Vol 18, pages: 651-656, 1975.

[11] J. Gaschnig, "Performance measurement and analysis of certain search algorithms", Techical Repot CMU-CS, pages: 79-124, Carnegie-Mellon University, 1979.

[12] P. Prosser, "Hybrid algorithms for the constraint satisfaction problem", Computational Intelligence, Vol 9, pages: 268-299, August 1993.

[13] D. Frost and R. Dechter, "Dead-End Driven Learning", Proceedings of the Twelfth National Conference of Artificial Intelligence, pages: 294-300, 1994

[14] M. J. Dent and R. E. Mercer, "Minimal Forward Checking", In Proc. Of the 6th International Conference on Tools with Artificial Intelligence, pages: 432-438, 1994.

[15] R. Haralick and G. Elliot, "Increasing tree efficiency for constraint satisfaction problems", Artificial Intelligence Vol 14, pages: 263-314, 1980.

[16] R. Dechter and I. Meiri., "Experimental evaluation of preprocessing algorithms for constraints satisfaction problems", Artificial Intelligence Vol 68, pages: 211-241, 1994.

[17] C. Schulte and G. Smolka, "Finite Domain of Constraints Programming in Oz. A Tutorial", Version 1.4.0, Mozart-oz.org, 2008.

[18] S. Haridi and N. Framzén,"Tutorial of Oz", Version 1.4.0, Mozart-oz.org , 2008.

[19] G. Dantzig, R. Fulkerson and S. Johnson, "Solution of a large-scale traveling-salesman problem", pages: 393-410, 1954.

[20] Frederick S. and Hillier, "Introducción a la investigación de operaciones", 8ª Ed, Editorial: McGraw-Hill, 2006.

[21] Avriel, Mordecai, "Nonlinear Programming: Analysis and Methods", Dover Publishing ISBN 0-486-43227-0, 2003

[22] M. Held y R. Karp "A Dynamic Programming Approach to Sequencing Problems" Journal of the Society for Industrial and Applied Mathematics, Vol. 10, No. 1 pages: 196-210, 1962.

[23] R. Companys Pascual y M. Mateo Doll "Un Algoritmo branch-and-bound doble para resolver el problema flow-shop con bloqueos" Dpto. de Organización de Empresas, Escuela Politécnica Superior de Ingeniería Industrial de Barcelona, IX Congreso de Ingeniería de Organización Gijón Paginas 1-12, 2005.

[24] G. Dantzig, R. Fulkerson y S. Johnson, "Using cutting planes to solve the symmetric traveling salesman problem", Mathematical Programming ISSN-1436-4646 Vol 15 pages: 177-188 http://www.tsp.gatech.edu/methods/dfj/index.html , 1978.

[25] Edward Tsang "Foundations of Constraint Satisfaction", Department of Computer Science University of Essex Colchester, Essex UK, ISBM 0-12-701610-4, pages: 299 – 320, 1993.

[26] J. N. Hooker "Combinando Optimización con Programación de Restricciones", Carniegie Mellon University, PPT Slides , Marzo del 2000.

[27] Franz Nauck. Schach. Illustrierter Zeitung, pages: 361-352, 1850.

[28] J.W.L Glaisher. Philosophical Magazine, Vol 18 pages: 457-467, 1874.

[29] I.Vardi I.Rivin and P. Zimmermann. "The n-queens problem". The American Mathermatical Monthy Vol 101, Pages: 629-639, 1994.

[30] K.D. Crawfor. "Solving the n-queens problem using genetic algorithms",, In Procedings ACM/SIGAPP Symposium on Applied Computing, Kansas City, pages: 1039-1047, 1992.

[31] Rok Sosic and Jun Gu. "Efficient local search with conflict minimization: A case study of the n-queens problem." Knowledfe and Data Engineering, Vol 6 pages: 661-668 1994.

[32] H. S. Stone and J. M. Stone. "Efficient local search with conflict minimization: A case study of the n-queens problem." IBM J. Res. Develop, Vol 30 pages: 242-258. 1986.

[33] L.V. Kalé. "An almost perfect heuristic for the n nonattacking queens problem." Information Processing Lettes, Vol 66 pages: 375-379, 1992".

[34] Osvaldo Clúa. "Técnicas de Programación Concurrente II: Mozart-oz." Facultad de Ingeniería, Universidad de Buenos Aires (FIUBA) paginas 62-75, 2004".

[35] Rubén Hinojosa Chapel. "La Composición Algorítmica como un Problema de Satisfacción de Restricciones" Universidad Pompeu Fabra, Octa 1, 08003 Barcelona España, paginas: 1-10, 2005".

[36] María Teresa Abad Soriano. "Algoritmos Voraces" Departament Llenguatges i Sistems Informàtics, Universitat Politècnica de Catalunya UPC, paginas: 3-5, 2007.

[37] Site Web. "DNIT: Departamento Nacional de Infra-Estructura de Transportes" Ministério dos Transportes Brasil, www1.dnit.gov.br 2009.

[37] Site Web. "Travel Notes™ - The Online Guide to Travel", www.travelnotes.org 2009.

[38] Victor Peña y Lillo Zumelzu. "Fundamentos de Investigación de Operaciones: Asignación y Vendedor Viajero", Escuela Ingeniería Informática, Universidad Técnica Federico Santa María, paginas: 10-12, 2009.

[39] C. S. Helvig, Gabriel Robins, and Alex Zelikovsky. “Moving-Target TSP and Related Problems”, Department of Computer Science, University of Virginia, Charlottesville, pages: 1-12, 1998.

[40] Reingold E., Nievergelt J., Deo N., “Combinatorial Algorithms: Theory and Practice”, Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1977.

[41] Bernardo Feijoo Péres., “Programación Entera”, Departamento de Cómputo Científico y Estadística, Universidad Simón Bolívar, Venezuela, páginas: 78-88, 2006.

[42] Sitio web “Vialidad del Gobierno de Chile”, “Mejorar la conectividad entre los chilenos y chilenas y entre Chile”, www.vialidad.gov.cl, 2010.

Anexo

1. Manual de Usuario

En esta sección se explicará la forma de instalar la herramienta Mozart/Oz y su intérprete Emacs para así luego ser ejecutados. Luego se explicará sobre el manejo del prototipo de Sistema elaborado en el presente Proyecto y sus funcionalidades, este prototipo de sistema viene en el CD adjunto al Presente informe.

COMO INSTALAR MOZART/OZ

Para empezar existen 2 formas de instalar este programa, las cuales son las siguientes:

- Bajar el archivo ejecutable desde la página oficial de Mozart/Oz. http://www.mozart-oz.org/download/view.cgi

- Ejecutar el archivo que viene en el CD adjunto al presente proyecto, X:\Tesis Jonathan Vivanco Navarro\Programa\Mozart-1.4.0.20080704.exe

Para ambas opciones se siguen los siguientes pasos al ejecutar el archivo de instalación:

Elegir el idioma y hacer clip en el botón “Ok”, aparecerá la ventana que se muestra en la Figura 10.1, hacer clip en el botón “Next”.

Figura 10.1: Interfaz de instalación de Mozart/Oz

Acepte los términos de acuerdo de uso y luego prosiga haciendo clip en el botón “Ok” hasta que aparezca la ventana que se muestra en la Figura 10.2. Por último hacer clip en el botón “finish” para terminar la instalación del programa.

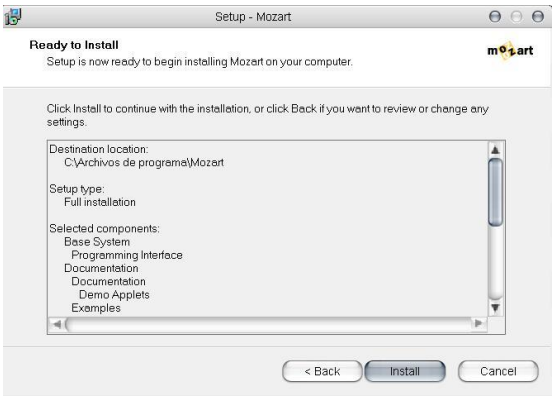

Figura 10.2 : Confirmación de Instalar de Mozart/Oz

COMO INSTALAR EMACS

Existen 2 formas de instalar este programa, las cuales son:

- Bajar el archivo ejecutable desde la página oficial. http://ftp.gnu.org/gnu/emacs/ la versión utilizada en este proyecto es la 20.7

- Ejecutar el archivo que viene en el CD adjunto al presente proyecto, X:\Tesis Jonathan Vivanco Navarro\Programa\emacs-20.7.exe

Para ambas opciones se siguen los siguientes pasos al ejecutar el archivo de instalación:

Ejecute el archivo de instalación y aparecerá la ventana que se muestra en la Figura 10.3, hacer clip en el botón "Next" hasta que le aparezca la confirmación de Instalar, luego acepte y aparecerá la ventana de la Figura 10.4. Finalmente presionar "finish" para terminar la instalación.

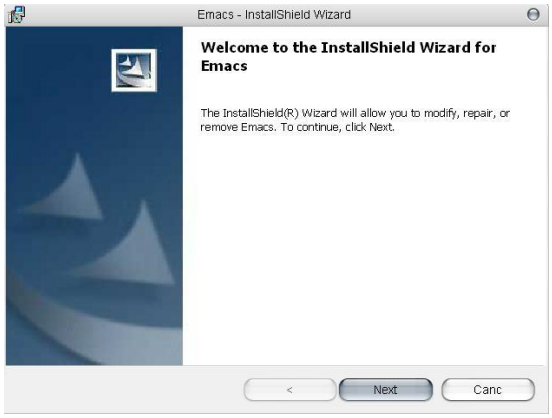

Figura 10.3: Instalación de Emacs

Figura 10.4: Confirmación de la Instalación de Emacs

COMO EJECUTAR EL PROTOTIPO DE SISTEMA

Antes de Ejecutar el Prototipo de Sistema, copia la carpeta TSP ubicada en X:\TSP, incluido en el CD adjunto al presente informe, a C:\ de su PC.

Para empezar ejecute el archivo "Prototipo Sistema Final.oz" incluido en el CD adjunto al presente informe, ubicado en X:\Tesis Jonathan Vivanco Navarro\ Prototipo de Sistema\Prototipo Sistema v Final.oz, luego prosiga con los siguientes pasos:

La primera ventana en abrirse es como se muestra en la Figura 10.5, selecciones en el menú la opción "OZ" y posteriormente "Feed Buffer" como se muestra en la Figura 10.6.

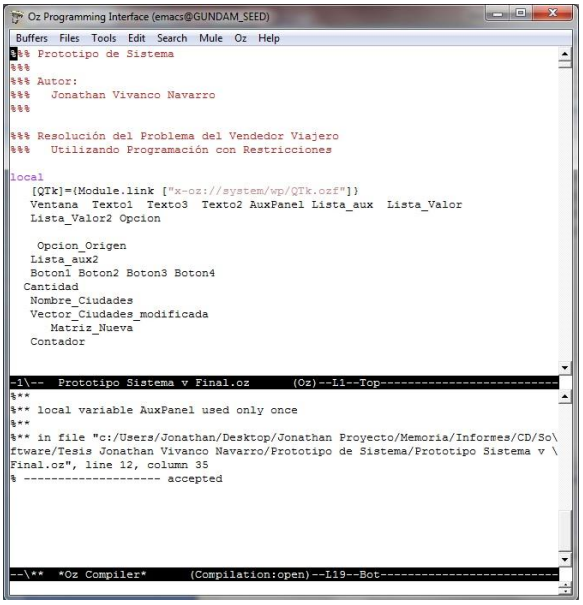

Figura 10.5: Interfaz previo a ejecutar El Prototipo de Sistema

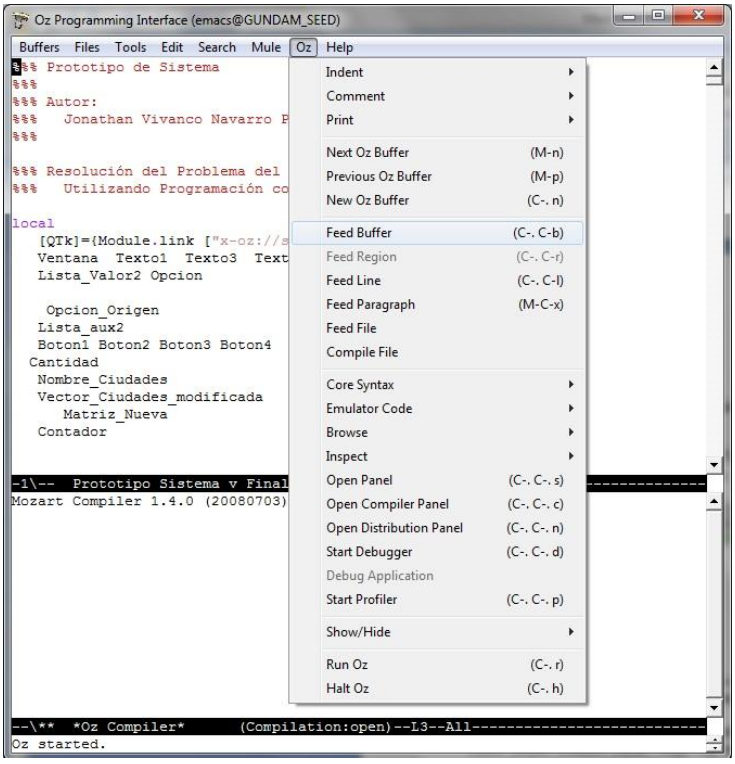

Figura 10.6: Ejecución de Prototipo de Sistema

Se abrirá la ventana mostrada en la Figura 10.7, la cual es la Interfaz del Prototipo de Sistema elaborado en el presente proyecto.

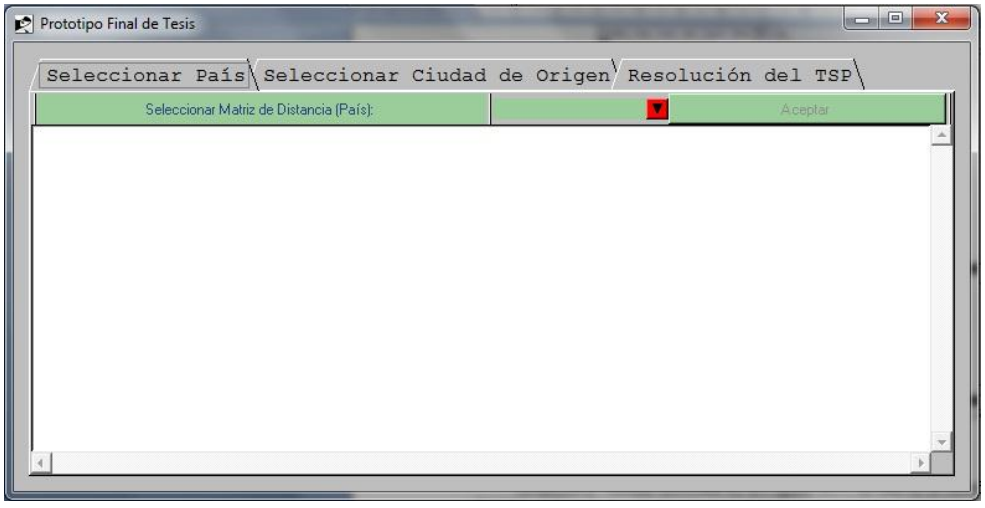

Figura 10.7: Interfaz del Prototipo de Sistema

En el Panel Inicio seleccione la matriz de distancia de uno de los siguientes países: Chile, Brasil, Estados Unido. Haga clip en el Botón "Aceptar", con esto se mostrarán los datos de distancias entre las ciudades del país seleccionado. A continuación haga clip en el panel "Seleccionar Origen" y seleccione la Ciudad de Origen para Calcular el TSP del país

seleccionado anteriormente, presionar “Aceptar”. Los datos mostrados serán parecidos a lo mostrado en la Figura 10.8.

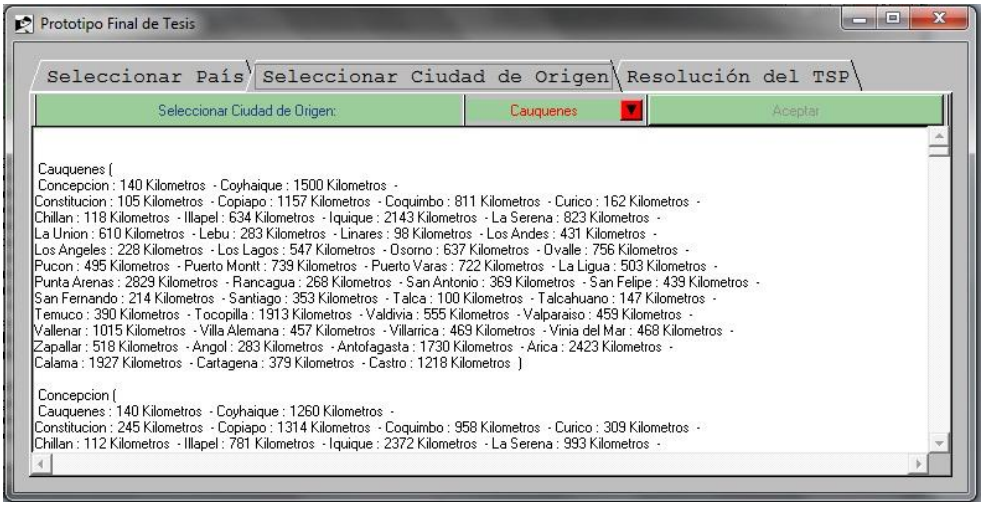

Figura 10.8: Panel Seleccionar Origen

Finalmente haga clip en el panel “Resolución del TSP” y presione “Calcular TSP”. El resultado se mostrara como en la Figura 10.9. Luego para terminar tiene 2 opciones:

1. Guardar el Resultado en un archivo plano: esta opción se logra presionando “Guardar” y luego en “Salir” finaliza la aplicación.

2. O simplemente presionando “Salir” para finalizar la aplicación, saliendo sin guardar el resultado del TSP.

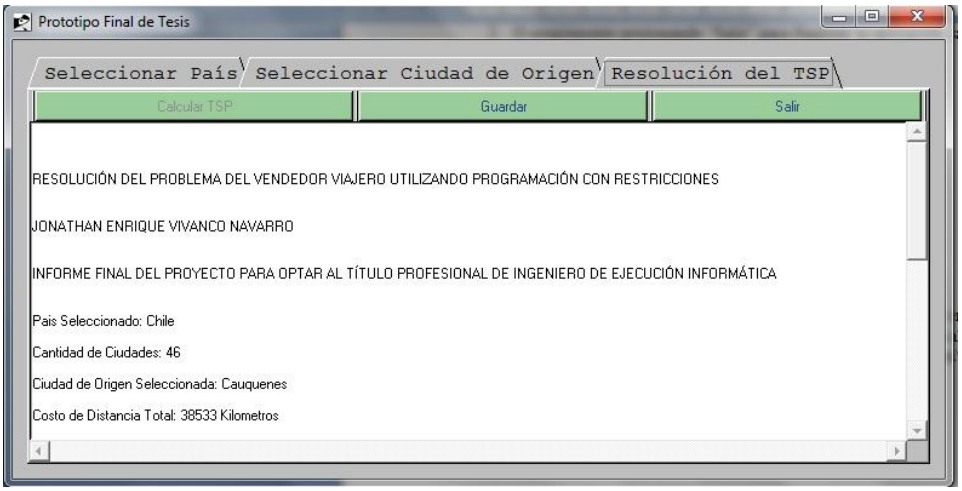

Figura 10.9: Panel Resolución del TSP

ACTUALIZAR BASE DE DATOS DEL PROTOTIPO

Para poder actualizar el contenido de la Carpeta TSP copiada en C:\, es necesario ejecutar el archivo "Tsp con CSOP v Final" incluido en el CD adjunto al presente informe, ubicado en X:\Tesis Jonathan Vivanco Navarro\ Prototipo de Sistema\ Tsp con CSOP v Final.oz, luego prosiga con los siguientes pasos:

La primera ventana en abrirse es la que se muestra en la Figura 10.5, seleccionar en el menú la opción "OZ" y posteriormente "Feed Buffer", de la misma forma que se muestra en la Figura 10.6. Para cambiar el país de origen y las ciudades, solo tiene que ir a modificar en el principio del código donde se encuentra escrito:

País="Estados Unidos" Opciones: Chile o Brasil o Estados Unidos y **Origen= 52** Numero según la ciudad elegida, estas se muestras en las tablas: 10.1, 10.2 y 10.3, luego tendrá que esperar 1 minuto para que se genere el TSP mediante CSOP.

Tabla 10.1: Ciudades de Chile utilizadas

Chile				
1. Angol	12. Coquimbo	23. Los Lagos	34. San Fernando	45. Viña del Mar
2. Antofagasta	13. Curico	24. Osorno	35. Santiago	46. Zapallar
3. Arica	14. Chillán	25. Ovalle	36. Talca	
4. Calama	15. Illapel	26. Pucón	37. Talcahuano	
5. Cartagena	16. Iquique	27. Puerto Montt	38. Temuco	
6. Castro	17. La Serena	28. Puerto Varas	39. Tocopilla	
7. Cauquenes	18. La Unión	29. La Ligua	40. Valdivia	
8. Concepción	19. Lebu	30. Punta Arenas	41. Valparaíso	
9. Coyhaique	20. Linares	31. Rancagua	42. Vallenar	
10. Constitución	21. Los Andes	32. San Antonio	43. Villa Alemana	
11. Copiapó	22. Los Ángeles	33. San Felipe	44. Villarica	

Tabla 10.2: Ciudades de Brasil utilizadas

Brasil		
1. Brasilia	13. Fortaleza	25. Recife
2. Aracaju	14. Goiãnia	26. Ribeirão Preto
3. Belém	15. João Pessoa	27. Rio Branco
4. Belo Horizonte	16. Joinville	28. Rio de Janeiro
5. Boa Vista	17. Juiz de Fora	29. Salvador
6. Campinas	18. Londrina	30. Santos
7. Campo Grande	19. Maceió	31. São José dos Campos
8. Caxias do Sul	20. Manaus	32. São Luís
9. Cuiabá	21. Natal	33. São Paulo
10. Curitiba	22. Pelotas	34. Teresina
11. Feira de Santana	23. Porto Alegre	35. Uberlãndia
12. Florianópolis	24. Porto Velho	36. Vitória

Tabla 10.3: Ciudades de Estados Unidos utilizadas

Estados Unidos		
1. Washington, DC	19. Denver, Colorado	37. Milwaukke, Wisconsin
2. Albany, New York	20. Des Moines, Iowa	38. Minneapolis, Minnesota
3. Atlanta, Georgia	21. Detroit, Michigan	39. Nashville, Tennessee
4. Albuquerque, New Mexico	22. El Paso, Texas	40. New Orleans, Louisiana
5. Augusta, Maine	23. Fargo, North Dakota	41. New York City, New York
6. Baltimore, Maryland	24. Grand Junction, Colorado	42. Norfolk, Virginia
7. Billings, Montana	25. Hartford, Connecticut	43. Oklahoma City, Oklahoma
8. Birmingham, Alabama	26. Houston, Texas	44. Omaha, Nebraska
9. Boise, Idaho	27. Indianapolis, Indiana	45. Orlando, Florida
10. Boston, Massachusetts	28. Jackson, Mississippi	46. Phidadelphia, Pennsylvania
11. Buffalo, New York	29. Jacksonville, Florida	47. Phoenix, Arizona
12. Charleston, South Carolina	30. Kansas City, Kansas/Missouri	48. Pittsburgh, Pennsylvania
13. Cheyenne, Wyoming	31. Las Vegas, Nevada	49. Reno, Nevada
14. Chicago, Illinois	32. Little Rock, Arkansas	50. San Antonio, Texas
15. Cheveland, Ohio	33. Los Ángeles, California	51. San Diego, California
16. Columbia, South Calorina	34. Louisville, Kentucky	52. San Francisco, California
17. Columbus, Ohio	35. Memphis, Tennessee	
18. Dallas/Ft Worth, Texas	36. Miami, Florida	

2. Código del Problema de Coloración de Mapas en Oz

Archivo mapas.oz

```
%% Proyecto 2 Jonathan Vivanco Navarro

declare

Data = [ los_andes              # [san_felipe_de_aconcagua]
       petorca                # [san_felipe_de_aconcagua quillota valparaiso]
       quillota               # [san_felipe_de_aconcagua petorca valparaiso]
       san_antonio            # [valparaiso quillota petorca]
       san_felipe_de_aconcagua  # [petorca quillota los_andes]
       valparaiso             # [san_antonio quillota petorca ] ]

fun {Unique Xs}
  case Xs of X1|X2|Xr then
    if X1==X2 then {Unique X2|Xr} else X1|{Unique X2|Xr} end
  else Xs
  end
end

fun {MapColoring Data N}
  Provincias = {Unique
                {Sort
                 {FoldR Data fun {$ C#Cs A} {Append Cs C|A} end nil}
                 Value.'<'}}
in
  proc {$ Color}

    Colores  = {FD.int 0#N}
  in
    {FD.distribute generic(order: fun{$ X Y}
                                    if{FD.reflect.size X}>{FD.reflect.size Y} then
                                        true
                                    else
                                        false
                                    end
                                  end

                           value: fun{$ S}

                                                {FD.reflect.nextLarger S {FD.reflect.mid S}}
                           end) [Colores]}
%%{FD.distribute naive [Colores]}
    %% Color: Provincias --> 1#colores

    {FD.record color Provincias 1#Colores Color} %% Esto crea implicitamente las variables para las provincias
    {ForAll Data
     proc {$ A#Bs}
       {ForAll Bs proc {$ B} Color.A \=: Color.B end}  %% se crea un propagador
     end}
    {FD.distribute generic(order: fun{$ X Y}
                                    if{FD.reflect.size X}>{FD.reflect.size Y} then
```

```
                                        true
                                    else
                                        false
                                    end
                                end
    value: fun{$ S}
                                                {FD.reflect.nextLarger S {FD.reflect.mid S}}
                            end) Color}
  end
end

{ExploreAll {MapColoring Data 5}}
```

3. Código del problema de N-Reinas en Oz

Archivo queens.oz

```
%% Proyecto 2 Jonathan Vivanco Navarro

declare
   fun {Queens N}
      proc {$ Row}
         L1N ={MakeTuple c N}
         LM1N={MakeTuple c N} %% Row a una tupla con un componente de cada reina. Esto crea implícitamente las
variables
      in
         {FD.tuple queens N 1#N Row}
         {For 1 N 1 proc {$ I}
                  L1N.I=I LM1N.I=~I
               end}
         {FD.distinct Row}
         {FD.distinctOffset Row LM1N}
         {FD.distinctOffset Row L1N}
             {FD.distribute generic(order: size
                                          value: fun {$ S}
                                          {FD.reflect.nextLarger S{FD.reflect.mid S}}
                                         end) Row}

      end
   end

   {ExploreOne {Queens 20}}
```

4. Código del Prototipo de Sistema en Oz

```
%%%
%%% Autor:
%%%   Jonathan Vivanco Navarro
%%%
%%% Resolucion del Problema del Vendedor Viajero
%%%   Utilizando Programación con Restricciones
%%%
declare
%%%
%%%%%%VARIABLES DE OPCIÖN

Pais="Estados Unidos"  %%%Opciones: Chile o Brasil o Estados Unidos
Origen= 52   %Numero segun el orden de las ciudades mostrado en la tabla del informe y en el manual de usaurio

%%%%%%%%%%%%%%%%%%%%
%%%%CHILE%%%%%%%%%% 46 ciudades
Matriz = distancia(

        angol(9999 1946 2638 2143 594 649 283 147 1241 374 1372 1027 378 166 849 2358 1038 351 161 270 646 60
288 378 972 237 480 464 719 2571 483 585 655 429 568 316 161 131 2129 297 674 1231 673 211 683 733)
        antofagasta(1946 9999 701 215 1391 2552 1730 1880 3011 1736 566 918 1571 1768 1184 492 887 2272 2020
1684 1337 1878 2210 2274 992 2158 2377 2385 1234 4451 1448 1397 1321 1516 1361 1618 1884 2038 185 2202 1313 722
1313 2132 1319 1250)
        arica(2638 701 9999 593 2084 3253 2423 2581 3712 2428 1261 1611 2263 2489 1876 316 1602 2965 2713 2376
2030 2579 2902 2975 1685 2850 3078 3078 1926 5152 2149 2030 2014 2209 2062 2319 2577 2739 536 2903 2020 1414
2006 2825 2011 1943)
        calama(594 215 593 9999 1589 2767 1927 2074 3360 1933 771 1116 1768 1978 1381 380 1106 2469 2217 1881
1534 2087 2407 2497 1189 2355 2598 2582 1431 4689 1657 1594 1518 1713 1574 1831 2081 2249 156 2415 1525 919
1510 2329 1516 1448)
        cartagena(584 1391 2084 1589 9999 1218 379 525 1811 384 818 472 219 429 295 1804 484 921 668 332 178 539
858 948 417 806 1050 1033 165 3140 136  9  177 165 112 282 533 701 1575 866 79  677 78 780 88 148)
        castro(649 2552 3253 2767 1218 9999 1218 801 414  999 1966 1651 1002 776 1474 3036 1664 314 778 894 1270
676 361 284 1596 506 175 190 1343 1994 1104 1209 1279 1054 1191 934 805 514 2753 385 1310 1855 1297 480 1308
1358)
        cauquenes(283 1730 2423 1927 379 1218 9999 140 1500 105 1157 811 162 118 634 2143 823 610 283 98 431 228
547 637 756 495 739 722 503 2829 268 369 439 214 353 100 147 390 1913 555 459 1015 457 469 468 518)
        concepcion(147 1880 2581 2074 525 801 140 9999 1260 245 1314 958 309 112 781 2372 993 493 143 201 577
127 430 523 903 378 626 606 650 2700 432 516 586 361 519 262 16 267 2060 452 639 1162 604 353 615 665)
        coyhaique(1241 3011 3712 3360 1811 414 1500 1260 9999 1591 2451 2244 1595 1384 2067 3503 2255 907 1370
1487 1863 1275 954 869 2189 1098 764 782 1936 1434 1701 1802 1872 1646 1786 1533 1397 1112 3346 973 1891 2448
1890 1073 1900 1950)
        constitucion(374 1736 2428 1933 384  999 105 245 1591 9999 1162 817 168 209 639 2148 828 701 388 112 436
319 638 728 762 587 830 814 509 2921 273 375 445 219 358 106 253 481 1919 647 464 1021 463 561 473 523)
        copiapo(1372 566 1261 771 818 1992 1157 1314 2451 1162 9999 345 997 1207 610 986 336 1699 1447 1110 764
1317 1636 1726 419 1584 1828 1812 660 3919 887 824 748 943 803 1060 1311 1479 757 1644 754 148 740 1559 745 677)
        coquimbo(1027 918 1611 1116 472 1651 811 958 2244 817 345 9999 652 862 265 1331  11  1353 1101 765 418
971 1291 1381 94 1239 1482 1466 315 3573 514 478 402 597 458 715 965 1133 1102 1299 409 204 394 1213 400 332)
        curico(378 1571 2263 1768 219 1002 162 309 1595 168 997 652 9999 213 475 1983 664 704 443 116 271 322 642
732 597 590 833 817 344 2924 109 210 280 55 194 66 316 484 1754 650 300 856 298 564 309 359)
        chillan(166 1768 2489 1978 429 776 118 112 1384 209 1207 862 213 9999 684 2193 873 493 241 105 481 111
430 521 807 379 622 606 554 2713 318 420 490 264 403 151 105 273 1964 439 509 1066 508 353 518 568)
        illapel(849 1184 1876 1381 295 1474 634 781 2067 639 610 265 475 684 9999 1596 276 1176 924 587 241 794
1113 1204 184 1062 1305 1289 137 3396 364 301 225 420 280 538 788 956 1367 1122 232 469 217 1036 223 154)
        iquique(2358 492 316 380 1804 3036 2143 2372 3503 2148 986 1331 1983 2193 1596 9999 1322 2685 2432 2096
1750 2303 2622 2712 1405 2570 2814 2798 1646 4905 1873 1809 1734 1929 1798 2046 2297 2465 229 2630 1740 1134
1726 2545 1731 1663)
        la_serena(1038 887 1602 1106 484 1664 823 993 2255 828 336  11  664 873 276 1322 9999 1365 1113 776 430
983 1302 1393 85 1251 1494 1487 326 3585 553 490 414 609 469 726 977 1145 1092 1311 421 194 406 1225 412 343)
        la_union(351 2272 2965 2469 921 314 610 493 907 701 1699 1353 704 493 1176 2685 1365 9999 480 597 973
385 64 44 1298 208 145 129 1045 2236 810 911 981 756 895 643 507 221 2456 66 1001 1558 999 182 1010 1060)
        lebu(161 2020 2713 2217 668 778 283 143 1370 388 1447 1101 443 241 924 2432 1113 480 9999 344 720 216
417 507 1046 365 609 593 793 2700 558 659 729 504 643 390 155 260 2203 425 749 1305 747 340 758 808)
```

```
            linares(270 1684 2376 1881 332 894 98 201 1487 112 1110 765 116 105 587 2096 776 597 344 9999 384 215 534
624 710 482 725 709 457 2816 221 323 393 167 306 54 209 377 1867 542 412 969 411 456 421 471)
            los_andes(646 1337 2030 1534 178 1270 431 577 1863 436 764 418 271 481 241 1750 430 973 720 384 9999 591
910 1000 363 858 1102 1085 110 3192 161 188 20 217 78 334 585 753 1521 918 129 623 114 832 120 125)
            los_angeles(60 1878 2579 2087 539 676  228 127 1275 319 1317 971 322 111 794 2303 983 385 216 215 591
9999 322 412 916 270 513 497 663 2604 428 529 600 374 513 261 153 165 2074 330 619 1176 618 244 628 678)
            los_lagos(288 2210 2902 2407 858 361 547 430 954 638 1636 1291 642 430 1113 2622 1302 64 417 534 910 322
9999 91 1236 145 193 176 983 2283 747 849 919 693 832 580 444 158 2393 67 938 1495 937 119 947 997)
            osorno(378 2274 2975 2497 948 284 637 523 869 728 1726 1381 732 521 1204 2712 1393 44 507 624 1000 412
91 9999 1326 235 107 91 1073 2199 838 939 1009 784 923 670 534 249 2483 110 1029 1585 1057 210 1038 1088)
            ovalle(972 992 1685 1189 417 1596 756 903 2189 762 419 94 597 807 184 1405 85 1298 1046 710 363 916 1236
1326 9999 1184 1427 1411 260 3518 486 423 348 542 403 660 910 1078 1176 1244 354 278 339 1158 345 277)
            pucon(237 2158 2850 2355 806 506 495 378 1098 587 1584 1239 590 379 1062 2570 1251 208 365 482 858 270
145 235 1184 9999 337 321 931 2428 696 797 867 642 781 528 392 107 2341 153 887 1443 885 26 896 946)
            puerto_montt(480 2377 3078 2598 1050 175 739 626 764 830 1828 1482 833 622 1305 2814 1494 145 609 725
1102 513 193 107 1427 337 9999 21 1174 2300 939 1040 1110 885 1024 771 636 350 2585 212 1130 1686 1128 311 1139
1189)
            puerto_varas(464 2385 3078 2582 1033 190 722 606 782 814 1812 1466 817 606 1289 2798 1487 129 593 709
1085 497 176 91 1411 321 21 9999 1158 2284 923 1024 1094 869 1008 755 620 334 2568 196 1114 1670 1112 295 1123
1173)
            la_ligua(719 1234 1926 1431 165 1343 503 650 1936 509 660 315 344 554 137 1646 326 1045 793 457 110 663
983 1073 260 931 1174 1158 9999 3265 233 170 94 289 150 407 657 825 1417 991 101 519 86 905 92 34)
            punta_arenas(2571 4451 5152 4689 3140 1994 2829 2700 1434 2921 3919 3573 2924 2713 3396 4905 3585 2236
2700 2816 3192 2604 2283 2199 3518 2428 2300 2284 3265 9999 3030 3131 3201 2976 3115 2862 2727 2441 4675 2303
3221 3777 3219 2402 3230 3280)
            rancagua(483 1448 2149 1657 136 1104 268 432 1701 273 887 514 109 318 364 1873 553 810 558 221 161 428
747 838 486 696 939 923 233 3030 9999 136 169 54 83 172 422 590 1644 756 189 746 188 670 198 248)
           san_antonio(585 1397 2030 1594  9  1209 369 516 1802 375 824 478 210 420 301 1809 490 911 659 323 188 529
849 939 423 797 1040 1024 170 3131 136 9999 197 155 112 273 523 691 1580 857 85 682 83 771 94 154)
            san_felipe(655 1321 2014 1518 177 1279 439 586 1872 445 748 402 280 490 225 1734 414 981 729 393 20 600
919 1009 347 867 1110 1094 94 3201 169 197 9999 225 87 343 593 761 1505 927 113 607 98 841 104 109)
            san_fernando(429 1516 2209 1713 165 1054 214 361 1646 219 943 597 55 264 420 1929 609 756 504 167 217
374 693 784 542 642 885 869 289 2976 54 155 225 9999 139 117 368 536 1700 702 245 801 243 616 254 304)
            santiago(568 1377 2070 1574 112 1191 353 519 1786 358 803 458 194 403 280 1798 469 895 643 306 78 513 832
923 403 781 1024 1008 150 3115 83 112 87 139 9999 257 507 675 1560 841 116 662 107 755 125 164)
            talca(316 1618 2319 1831 282 934 100 262 1533 106 1060 715 66 151 538 2046 726 643 390 54 334 261 580 670
660 528 771 755 407 2862 172 273 343 117 257 9999 255 423 1817 588 363 919 361 502 372 422)
           talcahuano(161 1884 2577 2081 533 805 147 16 1397 253 1311 965 316 105 788 2297 977 507 155 209 585 153
444 534 910 392 636 620 657 2727 422 523 593 368 507 255 9999 287 2068 452 613 1170 611 367 622 672)
           temuco(131 2038 2739 2249 701 514 390 267 1112 481 1479 1133 484 273 956 2465 1145 221 260 377 753 165
158 249 1078 107 350 334 825 2441 590 691 761 536 675 423 287 9999 2236 167 781 1338 779 81 790 840)
            tocopilla(2129 185 536 156 1575 2753 1913 2060 3346 1919 757 1102 1754 1964 1367 229 1092 2456 2203 1867
1521 2074 2393 2483 1176 2341 2585 2568 1417 4675 1644 1580 1505 1700 1560 1817 2068 2236 9999 2401 1511 905
1496 2315 1502 1434)
            valdivia(297 2202 2903 2415 866 385 555 452 973 647 1644 1299 650 439 1122 2630 1311 66 425 542 918 330
67 110 1244 153 212 196 991 2303 756 857 927 702 841 588 452 167 2401 9999 947 1503 945 128 956 1006)
            valparaiso(674 1319 2020 1525  79 1310 459 639 1891 464 754 409 300 509 232 1740 421 1001 749 412 129 619
938 1029 354 887 1130 1114 101 3221 189 85 113 245 116 363 613 781 1511 947 9999 613 25 861  8  72)
            vallenar(1231 722 1414 919 677 1855 1015 1162 2448 1021 148 204 856 1066 469 1134 194 1558 1305 969 623
1176 1495 1585 278 1443 1686 1670 519 3777 746 682 607 801 662 919 1170 1338 905 1503 613 9999 598 1417 604 536)
            villa_alemana(673 1313 2006 1510 78 1297 457 604 1890 463 740 394 298 508 217 1726 406 999 747 411 114
618 937 1057 339 885 1128 1112 86 3219 188 83 98 243 107 361 611 779 1496 945 25 598 9999 859 20 80)
            villarrica(211 2132 2825 2329 780 480 469 353 1073 561 1559 1213 564 353 1036 2545 1225 182 340 456 832
244 119 210 1158 26 311 295 905 2402 670 771 841 616 755 502 367 81 2315 128 861 1417 859 9999 870 920)
            vina_del_mar(683 1319 2011 1516 88 1308 468 615 1900 473 745 400 309 518 223 1731 412 1010 758 421 120
628 947 1038 345 896 1139 1123 92 3230 198 94 104 254 125 372 622 790 1502 956  8  604 20 870 9999 63)
             zapallar(733 1250 1943 1448 148 1358 518 665 1950 523 677 332 359 568 154 1663 343 1060 808 471 125 678
997 1088 277 946 1189 1173 34 3280 248 154 109 304 164 422 672 840 1434 1006 72 536 80 920 63 9999)
)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%Brasil%%%%%%%%%%%% 35 ciudades

MatrizB =distancias2(
```

```
brasilia(9999 1652 2120 716 4275 921 1134 1900 1133 1366 1415 1673 2378 209 2245 1503 992 1083 1928 3490 2422
2290 2027 2589 2220 706 3123 1148 1446 1087 1087 2157 1015 1789 435 1238)
aracaju(1652 9999 2079 1578 6000 2182 541 3169 2773 2595 322 2892 1183 1849 611 2722 1718 2584 294 5215 788 3559
3296 4229 501 2106 4763 1855 356 2249 2086 1578 2188 1142 2137 1408)
belem(2120 2079 9999 2824 6083 2842 2942 3727 2941 3193 1984 3500 1611 2017 2161 3330 3100 2891 2173 5298 2108
4117 3854 4397 2074 2622 4931 3250 2100 3005 3008 806 2933 947 2367 3108)
belo_horizonte(716 1578 2824 9999 4736 601 1453 1585 1594 1004 1256 1301 2528 906 2171 1131 272 1003 1854 3951
2348 1975 1712 3050 2061 523 3584 434 1372 658 611 2738 586 2302 556 524)
boa_vista(4275 6000 6083 4736 9999 4665 3836 5355 3142 4821 5378 5128 6548 4076 6539 4958 4978 4445 6276 785
6770 5611 5348 1686 6483 4445 2230 5159 5749 4828 4833 6120 4756 6052 4190 5261)
campinas(921 2182 2842 601 4665 9999 1012 1057 1523 476 1866 773 3133 835 2775 600 541 526 2458 3880 2952 1440
1177 2979 2665 238 3513 511 1982 171 174 2879  99 2698 497 959)
campo_grande(1134 541 2942 1453 3836 1012 9999 1525 694 991 2537 1298 3407 935 3357 1128 1512 615 3040 3051
3537 1781 1518 2150 3247 930 2684 1444 2568 1086 1107 2979 1014 2911 894 1892)
caxias_do_sul(1900 3169 3727 1585 5355 1057 1525 9999 2213 584 2847 478 4115 1720 3762 606 1498 913 3445 4570
3939 394 150 3669 3652 1215 4203 1426 2963 1054 1089 3764 982 3677 1476 1874)
cuiaba(1133 2773 2941 1594 3142 1523 694 2213 9999 1679 2539 1986 3406 934 3366 1816 1836 1303 3049 2357 3543
2469 2206 1456 3256 1303 1990 2017 2567 1686 1689 2978 1614 2910 1048 2119)
feria_de_santana(1366 2595 3193 1004 4821 476 991 584 1679 9999 2269 300 3541 1186 3188 130 914 379 2871 4036
3365 974 711 3135 3078 681 3669 852 2385 480 515 3230 408 3143 942 1300)
florianopolis(1415 322 1984 1256 5378 1866 2537 2847 2539 2269 9999 2566 1273 1528 915 2396 1396 2262 598 4893
1092 3237 2974 3907 805 1784 4441 1533 116 1918 1764 1483 1846 1047 1816 1124)
fortaleza(1673 2892 3500 1301 5128 773 1298 478 1986 300 2566 9999 3838 1493 3485 180 1221 686 3168 4343 3662 739
476 3442 3375 988 3976 1144 2682 777 807 3537 705 3450 1249 1597)
goiania(2378 1183 1611 2528 6548 3133 3407 4115 3406 3541 1273 3838 9999 2482 688 3668 2668 3356 1075 5763 537
4505 4242 4865 800 2978 5396 2805 1389 3199 3036 1070 3127 634 2707 2397)
joao_pessoa(209 1849 2017 906 4076 835 935 1720 934 1186 1528 1493 2482 9999 2442 1323 1174 884 2105 3291 2619
2110 1847 2390 2332 615 2924 1338 1643 998 1005 2054 926 1986 360 1428)
joinville(2245 611 2161 2171 6539 2775 3357 3762 3366 3188 915 3485 688 2442 9999 3315 2311 3177 395 5808 185
4152 3710 4907 120 2699 5356 2448 949 2842 2679 1660 2770 1224 2730 2001)
juiz_de_fora(1503 2722 3330 1131 4958 600 1128 606 1816 130 2396 180 3668 1323 3315 9999 1046 516 2998 4173 3492
903 640 3272 3205 818 3806 974 2512 607 2679 3367 535 3280 1079 1427)
londrina(992 1718 3100 272 4978 541 1512 1498 1836 914 1396 1221 2668 1174 2311 1046 9999 1024 1994 4193 2488
1888 1625 3292 2201 676 3826 184 1512 578 415 2874 506 2438 798 519)
maceio(1083 2584 2891 1003 4445 526 615 913 1303 379 2262 686 3356 884 3177 516 1024 9999 2860 3660 3354 1303
1040 1958 3067 478 3293 953 2374 603 621 2928 538 2860 716 1406)
manaus(1928 294 2173 1854 6276 2458 3040 3445 3049 2871 598 3168 1075 2105 395 2998 1994 2860 9999 5491 572
3835 3572 4505 285 2382 5124 2131 632 2525 2362 1672 2453 1236 2413 1684)
natal(3490 5215 5298 3951 785 3880 3051 4570 2357 4036 4893 4343 5763 3291 5808 4173 4193 3660 5491 9999 5985
4826 4563 901 5698 3660 1445 4374 5009 4043 4046 5335 3971 5267 3405 4476)
pelotas(2422 788 2108 2348 6770 2952 3537 3939 3543 3365 1092 3662 537 2619 185 3492 2488 3354 572 5985 9999
4329 4066 4999 297 2876 5533 2625 1126 3019 2859 1607 2947 1171 2907 2178)
porto_alegre(2290 3559 4117 1975 5611 1440 1781 394 2469 974 3237 739 4505 2110 4152 903 1888 1303 3835 4826
4329 9999 271 3925 4042 1605 4459 1816 3353 1444 1479 4154 1372 4067 1866 2264)
porto_velho(2027 3296 3854 1712 5348 1177 1518 150 2206 711 2974 476 4242 1847 3710 640 1625 1040 3572 4563
4066 271 9999 3662 3779 1342 4196 1553 3127 1181 1216 3891 1109 3804 1603 2001)
recife(2589 4229 4397 3050 1686 2979 2150 3669 1456 3135 3907 3442 4865 2390 4907 3272 3292 1958 4505 901 4999
3925 3662 9999 4712 2759 544 3473 4023 3142 3145 4434 3070 4366 2504 3575)
ribeirao_preto(2220 501 2074 2061 6483 2665 3247 3652 3256 3078 805 3375 800 2332 120 3205 2201 3067 285 5698 297
4042 3779 4712 9999 2589 5243 2338 839 2732 2569 1573 2660 1137 2620 1891)
rio_branco(706 2106 2622 523 4445 238 930 1215 1303 681 1784 988 2978 615 2699 818 676 478 2382 3660 2876 1605
1342 2759 2589 9999 3293 725 1900 391 408 2659 319 2483 281 1048)
rio_de_janeiro(3123 4763 4931 3584 2230 3513 2684 4203 1990 3669 4441 3976 5396 2924 5356 3806 3826 3293 5124
1445 5533 4459 4196 544 5243 3293 9999 4007 4457 3676 3679 4968 3604 4900 3038 4109)
salvador(1148 1855 3250 434 5159 511 1444 1426 2017 852 1533 1144 2805 1338 2448 974 184 953 2131 4374 2625 1816
1553 3473 2338 725 4007 9999 1649 501 343 3015 429 2579 979 521)
santos(1446 356 2100 1372 5749 1982 2568 2963 2567 2385 116 2682 1389 1643 949 2512 1512 2374 632 5009 1126 3353
3127 4023 839 1900 4457 1649 9999 2034 1880 1599 1962 1163 1932 1202)
sao_jose_dos_campos(1087 2249 3005 658 4828 171 1086 1054 1686 480 1918 777 3199 998 2842 607 578 603 2525 4043
3019 1444 1181 3142 2732 391 3676 501 2034 9999 169 3042 72  2864 662 954)
sao_lourenço(1087 2086 3008 611 4833 174 1107 1089 1689 515 1764 807 3036 1005 2679 2679 415 621 2362 4046 2859
1479 1216 3145 2569 408 3679 343 1880 169 9999 3049 97  2810 669 791)
sao_luis(2157 1578 806 2738 6120 2879 2979 3764 2978 3230 1483 3537 1070 2054 1660 3367 2874 2928 1672 5335 1607
4154 3891 4434 1573 2659 4968 3015 1599 3042 3049 9999 2970 446 2404 2607)
sao_paulo(1015 2188 2933 586 4756  99 1014 982 1614 408 1846 705 3127 926 2770 535 506 538 2453 3971 2947 1372
1109 3070 2660 319 3604 429 1962 72  97  2970 9999 2792 590 882)
teresina(1789 1142 947 2302 6052 2698 2911 3677 2910 3143 1047 3450 634 1986 1224 3280 2438 2860 1236 5267 1171
4067 3804 4366 1137 2483 4900 2579 1163 2864 2810 446 2792 9999 2212 2171)
uberlandia(435 2137 2367 556 4190 497 894 1476 1048 942 1816 1249 2707 360 2730 1079 798 716 2413 3405 2907 1866
1603 2504 2620 281 3038 979 1932 662 669 2404 590 2212 9999 1081)
```

```
vitoria(1238 1408 3108 524 5261 959 1892 1874 2119 1300 1124 1597 2397 1428 2001 1427 519 1406 1684 4476 2178
2264 2001 3575 1891 1048 4109 521 1202 954 791 2607 882 2171 1081 9999)
)
%%%%%%%%%%%%%%%%%%%%
%%%%%%%%USA%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% 52 ciudades

MatrizUSA = distancias_usa(
washington__DC(9999 645 1000 2984 1161 64  3403 1193 3822 725 661 854 2645 1145 580 774 693 2113 2613 1709 838
3113 2145 3032 580 2209 935 1564 1161 1677 3903 709 4274 967 1371 1709 1306 1758 1064 1774 387 306 2145 1838
1371 209 3709 403 4145 2564 4193 4580)
albany__New_York(645 9999 1629 3290 709 548 3387 1725 4064 274 483 1419 2887 1322 774 1322 1032 2709 2951
1903 1048 3580 2403 3387 177 2855 1306 2177 1725 2096 4242 2193 4597 1387 1983 2258 1500 2016 1661 2322 258 822
2419 2113 1983 419 4016 725 4451 3177 4597 4806)
albuquerque__New_Mexico(1000 1629 9999 2258 2145 1048 2903 241 3580 1790 1467 467 2387 1145 1177 338 951 1322
2306 1548 1177 2322 2242 2677 1548 1274 854 645 500 1322 3193 887 3532 677 612 1064 1290 1806 403 774 1371 903
1338 1629 693 1209 2951 1096 3887 1613 3467 4000)
augusta__Maine(2984 3290 2258 9999 3967 3048 1596 2016 1516 3580 2855 2742 871 2113 2564 2613 2354 1032 709
1564 2516 435 2080 693 3387 1371 2048 1693 2645 1258 851 1419 1306 2080 1629 3177 2225 1967 1983 1935 3226 3048
871 1435 2806 3145 741 2629 1645 1177 1306 1790)
atlanta__Georgia(1161 709 2145 3967 9999 1080 3984 2435 4774 435 1177 1967 3580 2016 1403 1951 1629 3258 3613
2532 1709 4242 3048 4048 596 3419 1967 2822 2371 2774 4887 2855 5322 2016 2629 2951 2129 2645 2290 2919 822
1435 3113 3758 2613 951 4709 1387 5064 3693 5274 5451)
baltimore__Maryland(64  548 1048 3048 1080 9999 3096 1242 3887 693 596 919 2693 1161 580 838 693 2193 2645 1677
822 3193 2177 3064 516 2258 919 1629 1209 1725 3871 1677 4274 1032 1467 1758 1274 1790 1129 1838 322 370 3113
1854 1435 161 3726 403 4226 2629 4322 4548)
billings__Montana(3403 3387 2903 1596 3984 3096 9999 2871 983 3548 2774 3516 741 1983 2596 3322 2596 2290 887
1580 2451 2032 983 1080 3387 2564 2274 2774 3548 1677 1693 2290 2000 2467 2345 4064 1871 1338 2580 2951 3290
3355 1854 1467 3596 3193 1935 2677 1548 2451 2113 1919)
birmingham__Alabama(1193 1725 241 2016 2435 1242 2871 9999 3338 1983 1451 709 2290 1064 1177 580 919 1032
2145 1338 1225 2048 2096 2629 1742 1129 774 403 741 1177 2951 693 3306 580 419 1209 1193 1725 306 564 1580 1145
1161 1419 887 1403 2709 1274 3677 1387 3209 3822)
boise__Idaho(3822 4064 3580 1516 4774 3887 983 3338 9999 4338 3564 4016 1177 2758 3274 3855 3177 2564 1354 2193
3193 1935 1951 1016 4209 2935 2984 3290 3984 2290 1080 2855 1371 3113 2935 4629 2806 2387 3193 3467 4016 4064
2322 2000 4258 3919 1580 3435 693 2693 1580 1048)
boston__Massachusetts(725 274 1790 3580 435 693 3548 1983 4338 9999 758 1516 3096 1613 1064 1548 1290 2822 3226
2161 1290 3838 2661 3661 177 2951 1500 2354 1871 2322 4435 2322 4871 1548 2161 2451 1758 2242 1758 2435 338 935
2725 2371 2096 516 4306 919 4629 3258 4645 5048)
buffalo__New_York(661 483 1467 2855 1177 596 2774 1451 3564 758 9999 1500 2419 871 306 1290 532 2193 2500 1371
580 3177 1919 2887 677 2403 822 1806 1758 1629 3677 1709 4161 887 1483 2258 1016 1532 1161 2032 596 967 1967
1645 1935 580 3580 354 3951 2629 4080 4306)
charleston__South_Carolina(854 1419 467 2742 1967 919 3516 709 4016 1516 1500 9999 2758 1467 1177 177 1048 1758
2774 1935 1354 2742 2435 3161 1338 1693 1161 1080 387 1790 3677 1338 4016 938 1129 951 1645 2145 871 1177 1242
709 1838 2048 612 1064 3419 1048 4387 2064 3903 4548)
cheyenne__Wyoming(2645 2887 2387 871 3580 2693 741 2290 1177 3096 2419 2758 9999 1580 2145 2677 2096 1419 161
1032 2000 1274 1258 564 2984 1790 1742 2048 2822 1096 1371 1725 1806 1935 1838 3467 1613 1419 2032 2225 2822
2903 1112 806 3016 2838 1516 2306 1564 1709 1919 1919)
chicago__Illinois(1145 1322 1145 2113 2016 1161 1983 1064 2758 1613 871 1467 1580 9999 564 1290 580 1483 1645 548
451 2354 1048 2048 1467 1758 306 1209 1629 871 2871 1064 3306 483 871 2258 145 661 758 1483 1306 1403 1338 774
1854 1274 2806 774 3145 1951 3371 3500)
cheveland__Ohio(580 774 1177 2564 1403 580 2596 1177 3274 1064 306 1177 2145 564 9999 983 225 1919 2193 1064
274 2806 1613 2596 903 2113 516 1516 1467 1322 3371 1403 3838 564 1177 2016 709 1225 854 1709 758 822 1661 1338
1693 693 3274 209 3645 2338 3855 4000)
columbia__South_Calorina(774 1322 338 2613 1951 838 3322 580 3855 1548 1290 177 2677 1290 983 9999 854 1661
2613 1790 1161 2629 2306 2967 1306 1661 983 967 483 1645 3532 1225 3903 806 1000 1032 1419 1951 709 1112 1161
629 1709 1903 709 983 3274 951 4209 1983 3838 4468)
columbus__Ohio(693 1032 951 2354 1629 693 2596 919 3177 1290 532 1048 2096 580 225 854 9999 1693 2000 1064 306
2580 1532 2403 1064 1887 290 1290 1338 1096 3258 1177 3613 338 951 1887 758 1258 612 1483 903 903 1483 1274
1548 774 3064 306 3564 2113 3645 3919)
dallas_Ft_Worth__Texas(2113 2709 1322 1032 3258 2193 2290 1032 2564 2822 2193 1758 1419 1483 1919 1661 1693
9999 1258 1129 1871 1000 1758 1580 2725 403 1435 661 1677 822 1983 516 2258 1338 725 2161 1645 1532 1064 838
2516 2177 338 1064 1774 2322 1613 1951 2693 435 2177 2822)
denver__Colorado(2613 2951 2306 709 3613 2645 887 2145 1354 3226 2500 2774 161 1645 2193 2613 2000 1258 9999
1080 2064 1112 1419 403 3226 1661 1709 1935 2806 983 1225 1516 1661 1806 1677 3403 1677 1483 1903 2064 2887
2855 1016 871 3032 2806 1306 2306 1661 1532 1774 2032)
```

des_Moines__Iowa(1709 1903 1548 1564 2532 1677 1580 1338 2193 2161 1371 1935 1032 548 1064 1790 1064 1129 1080 9999 951 1822 774 1483 2016 1516 774 1338 2000 322 2306 903 2758 951 1000 2564 596 403 1112 1580 1806 1854 887 209 2193 1758 2306 1258 2580 1580 2855 2951)
detroit__Michigan(838 1048 1177 2516 1709 822 2451 1225 3193 1290 580 1354 2000 451 274 1161 306 1871 2064 951 9999 2693 1483 2451 1177 2064 451 1548 1693 1242 3258 1419 3693 612 1161 2242 580 1112 871 1725 1048 1177 1661 1177 1887 983 3242 483 3548 2338 3822 3871)
el_Paso__Texas(3113 3580 2322 435 4242 3193 2032 2048 1935 3838 3177 2742 1274 2354 2806 2629 2580 1000 1112 1822 2693 9999 2338 1080 3693 1193 2290 1661 2548 1516 1161 1516 1322 2338 1725 3064 2484 2354 2064 1774 3467 3145 1096 2016 2709 3338 709 2871 1887 903 1177 1935)
fargo__North_Dakota(2145 2403 2242 2080 3048 2177 983 2096 1951 2661 1919 2435 1258 1048 1613 2306 1532 1758 1419 774 1483 2338 9999 1790 2516 2096 1338 2113 2725 1000 2484 1677 2919 1516 1822 3226 919 387 1790 2225 2338 2371 1403 693 2919 2306 2725 1822 2548 2145 3016 2935)
grand_Junction__Colorado(3032 3387 2677 693 4048 3064 1080 2629 1016 3661 2887 3161 564 2048 2596 2967 2403 1580 403 1483 2451 1080 1790 9999 3467 1967 2145 2274 3161 1387 822 1854 1258 2209 2048 3709 2080 1871 2290 2371 3290 3258 1322 1274 3338 3209 935 2725 1209 1758 1371 1580)
hartford__Connecticut(580 177 1548 3387 596 516 3387 1742 4209 177 677 1338 2984 1467 903 1306 1064 2725 3226 2016 1177 3693 2516 3467 9999 2790 1354 2177 1693 2161 4290 2161 4677 1403 1935 2290 1613 2129 1613 2306 193 790 2645 2209 1774 338 4145 758 4516 3080 4677 4968)
houston__Texas(2209 2855 1274 1371 3419 2258 2564 1129 2935 2951 2403 1693 1790 1758 2113 1661 1887 403 1661 1516 2064 1193 2096 1967 2790 9999 1613 677 1435 1193 2371 693 2484 1516 919 1919 1903 1903 1258 564 2596 2177 741 1387 1580 2435 1871 2209 3032 322 2403 3080)
indianapolis__Indiana(935 1306 854 2048 1967 919 2274 774 2984 1500 822 1161 1742 306 516 983 290 1435 1709 774 451 2290 1338 2145 1354 1613 9999 1080 1354 806 2967 983 3322 177 758 1919 435 951 451 1290 1177 1145 1193 983 1564 1064 2790 580 3322 1919 3355 3693)
jackson__Mississippi(1564 2177 645 1693 2822 1629 2774 403 3290 2354 1806 1080 2048 1209 1516 967 1290 661 1935 1338 1548 1661 2113 2274 2177 677 1080 9999 983 1112 2629 419 2935 951 338 1467 1338 1709 661 338 1967 1532 903 1403 1129 1790 2354 1516 3355 1080 2838 3467)
jacksonville__Florida(1161 1725 500 2645 2371 1209 3548 741 3984 1871 1758 387 2822 1629 1467 483 1338 1677 2806 2000 1693 2548 2725 3161 1693 1435 1354 983 9999 1838 3613 1322 3790 1177 1112 564 1790 2354 903 903 1532 1000 1854 2129 225 1371 3226 1338 1338 1742 3806 4435)
kansas_City__Kansas_Missouri(1677 2096 1322 1258 2774 1725 1677 1177 2290 2322 1629 1790 1096 871 1322 1645 1096 822 983 322 1242 1516 1000 1387 2161 1193 806 1112 1838 9999 2209 677 2548 822 774 2387 903 709 935 1354 1983 1871 564 306 2016 1887 2000 1403 2645 1258 2564 3000)
las_Vegas__Nevada(3903 4242 3193 851 4887 3871 1693 2951 1080 4435 3677 3677 1371 2871 3371 3532 3258 1983 1225 2306 3258 1161 2484 822 4290 2371 2967 2629 3613 2209 9999 2354 435 3016 2580 4145 2903 2677 2919 2790 4145 4000 1790 2096 3790 4000 467 3564 725 2080 548 919)
little_Rock__Arkansas(709 2193 887 1419 2855 1677 2290 693 2855 2322 1709 1338 1725 1064 1403 1225 1177 516 1516 903 1419 1516 1677 1854 2161 693 983 419 1322 677 2354 9999 2725 822 225 1871 1242 1338 564 677 2016 1645 548 935 1548 1838 2145 1451 645 2693 3209 3629)
los_Angeles__California(4274 4597 3532 1306 5322 4274 2000 3306 1371 4871 4161 4016 1806 3306 3838 3903 3613 2258 1661 2758 3693 1322 2919 1258 4677 2484 3322 2935 3790 2548 435 2725 9999 3516 2919 4387 3338 3000 3242 3000 4500 4338 2161 2532 3919 4355 629 3919 758 2242 209 629)
louisville__Kentucky(967 1387 677 2080 2016 1032 2467 580 3113 1548 887 938 1935 483 564 806 338 1338 1806 951 612 2338 1516 2209 1403 1516 177 951 1177 822 3016 822 3516 9999 612 1758 612 1145 274 1129 1242 1048 1209 1129 1387 1145 2822 629 3451 1774 3371 3790)
memphis__Tennessee(1371 1983 612 1629 2629 1467 2345 419 2935 2161 1483 1129 1838 871 1177 1000 951 725 1677 1000 1161 1725 1822 2048 1935 919 758 338 1112 774 2580 225 2919 612 9999 1613 1000 1467 338 661 1774 1419 774 1032 1258 1629 2371 1225 3274 1177 2919 3419)
miami__Florida(1709 2258 1064 3177 2951 1758 4064 1209 4629 2451 2258 951 3467 2258 2016 1032 1887 2161 3403 2564 2242 3064 3226 3709 2290 1919 1919 1467 564 2387 4145 1871 4387 1758 1613 9999 2354 2855 1467 1387 2145 1564 2419 2645 370 1983 3790 1903 4829 2242 4322 4984)
milwaukke___Wisconsin(1306 1500 1290 2225 2129 1274 1871 1193 2806 1758 1016 1645 1613 145 709 1419 758 1645 1677 596 580 2484 919 2080 1613 1903 435 1338 1790 903 2903 1242 3338 612 1000 2354 9999 548 887 1661 1435 1532 1419 806 1967 1403 2855 903 3161 2080 3435 3500)
minneapolis__Minnesota(1758 2016 1806 1967 2645 1790 1338 1725 2387 2242 1532 2145 1419 661 1225 1951 1258 1532 1483 403 1112 2354 387 1871 2129 1903 951 1709 2354 709 2677 1338 3000 1145 1467 2855 548 9999 1338 2177 1967 1983 1274 612 2532 1935 2709 1435 2951 2016 3226 3193)
nashville__Tennessee(1064 1661 403 1983 2290 1129 2580 306 3193 1758 1161 871 2032 758 854 709 612 1064 1903 1112 871 2064 1790 2290 1613 1258 451 661 903 935 2919 564 3242 274 338 1467 887 1338 9999 854 1451 1080 1096 1225 1112 1274 2693 919 3580 1500 3226 3758)
new_Orleans__Louisiana(1774 2322 774 1935 2919 1838 2951 564 3467 2435 2032 1177 2225 1483 1709 1112 1483 838 2064 1580 1725 1774 2225 2371 2306 564 1290 338 903 1354 2790 677 3000 1129 661 1387 1661 2177 854 9999 2161 1677 1112 1661 1048 1983 2419 1838 3596 887 2967 3677)
new_York_City__New_York(387 258 1371 3226 822 322 3290 1580 4016 338 596 1242 2822 1306 758 1161 903 2516 2887 1806 1048 3467 2338 3290 193 2596 1177 1967 1532 1983 4145 2016 4500 1242 1774 2145 1435 1967 1451 2161 9999 596 2387 2016 1758 177 3951 612 4371 2935 4516 4726)
norfolk__Virginia(306 822 903 3048 1435 370 3355 1145 4064 935 967 709 2903 1403 822 629 903 2177 2855 1854 1177 3145 2371 3258 790 2177 1145 1532 1000 1871 4000 1645 4338 1048 1419 1564 1532 1983 1080 1677 596 9999 2209 2129 1242 435 3790 677 4500 2500 4322 4839)

```
oklahoma_City__Oklahoma(2145 2419 1338 871 3113 3113 1854 1161 2322 2725 1967 1838 1112 1338 1661 1709 1483
338 1016 887 1661 1096 1403 1322 2645 741 1193 903 1854 564 1790 548 2161 1209 774 2419 1419 1274 1096 1112
2387 2209 9999 741 1983 2242 1580 1790 2516 774 2145 2677)
omaha__Nebraska(1838 2113 1629 1435 3758 1854 1467 1419 2000 2371 1645 2048 806 774 1338 1903 1274 1064 871
209 1177 2016 693 1274 2209 1387 983 1403 2129 306 2096 935 2532 1129 1032 2645 806 612 1225 1661 2016 2129 741
9999 2274 1935 2177 1500 2371 1516 2629 2725)
orlando_Florida(1371 1983 693 2806 2613 1435 3596 887 4258 2096 1935 612 3016 1854 1693 709 1548 1774 3032 2193
1887 2709 2919 3338 1774 1580 1564 1129 225 2016 3790 1548 3919 1387 1258 370 1967 2532 1112 1048 1758 1242
1983 2274 9999 1596 3355 1580 4580 1887 3887 4629)
phidadelphia__Pennsylvania(209 419 1209 3145 951 161 3193 1403 3919 516 580 1064 2838 1274 693 983 774 2322 2806
1758 983 3338 2306 3209 338 2435 1064 1790 1371 1887 4000 1838 4355 1145 1629 1983 1403 1935 1274 1983 177 435
2242 1935 1596 9999 3822 500 4274 2806 4468 4677)
phoenix__Arizona(3709 4016 2951 741 4709 3726 1935 2709 1580 4306 3580 3419 1516 2806 3274 3274 3064 1613 1306
2306 3242 709 2725 935 4145 1871 2790 2354 3226 2000 467 2145 629 2822 2371 3790 2855 2709 2693 2419 3951 3790
1580 2177 3355 3822 9999 3371 1177 1613 564 1225)
pittsburgh__Pennsylvania(403 725 1096 2629 1387 403 2677 1274 3435 919 354 1048 2306 774 209 951 306 1951 2306
1258 483 2871 1822 2725 758 2209 580 1516 1338 1403 3564 1451 3919 629 1225 1903 903 1435 919 1838 612 677 1790
1500 1580 500 3371 9999 3822 2387 3935 4209)
reno__Nevada(4145 4451 3887 1645 5064 4226 1548 3677 693 4629 3951 4387 1564 3145 3645 4209 3564 2693 1661
2580 3548 1887 2548 1209 4516 3032 3322 3355 1338 2645 725 645 758 3451 3274 4829 3161 2951 3580 3596 4371 4500
2516 2371 4580 4274 1177 3822 9999 2790 967 370)
san_Antonio__Texas(2564 3177 1613 1177 3693 2629 2451 1387 2693 3258 2629 2064 1709 1951 2338 1983 2113 435
1532 1580 2338 903 2145 1758 3080 322 1919 1080 1742 1258 2080 2693 2242 1774 1177 2242 2080 2016 1500 887 2935
2500 774 1516 1887 2806 1613 2387 2790 9999 2096 2806)
san_Diego__California(4193 4597 3467 1306 5274 4322 2113 3209 1580 4645 4080 3903 1919 3371 3855 3838 3645 2177
1774 2855 3822 1177 3016 1371 4677 2403 3355 2838 3806 2564 548 3209 209 3371 2919 4322 3435 3226 3226 2967
4516 4322 2145 2629 3887 4468 564 3935 967 2096 9999 822)
san_Francisco__California(4580 4806 4000 1790 5451 4548 1919 3822 1048 5048 4306 4548 1919 3500 4000 4468 3919
2822 2032 2951 3871 1935 2935 1580 4968 3080 3693 3467 4435 3000 919 3629 629 3790 3419 4984 3500 3193 3758
3677 4726 4839 2677 2725 4629 4677 1225 4209 370 2806 822 9999)
)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Vector_Chile = ciudades("Angol" "Antofagasta" "Arica" "Calama" "Cartagena" "Castro" "Cauquenes" "Concepcion"
"Coyhaique" "Constitucion" "Copiapo" "Coquimbo" "Curico" "Chillan" "Illapel" "Iquique" "La Serena" "La Union" "Lebu"
"Linares" "Los Andes" "Los Angeles" "Los Lagos" "Osorno" "Ovalle" "Pucon" "Puerto Montt" "Puerto Varas" "La Ligua"
"Punta Arenas" "Rancagua" "San Antonio" "San Felipe" "San Fernando" "Santiago" "Talca" "Talcahuano" "Temuco"
"Tocopilla" "Valdivia" "Valparaiso" "Vallenar" "Villa Alemana" "Villarrica" "Vinia del Mar" "Zapallar")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Vector_Brasil = ciudad("Brasilia" "Aracaju" "Belem" "Belo Horizonte" "Boa Vista" "Campinas" "Campo Grande" "Caxias
Do Sul" "Cuiaba" "Curitiba" "Feria de Santana" "Florianopolis" "Fortaleza" "Goiania" "Joao Pessoa" "Joinville" "Juiz de
Fora" "Londrina" "Maceio" "Manaus" "Natal" "Pelotas" "Porto Alegre" "Porto Velho" "Recife" "Ribeirao Preto" "Rio
Branco" "Rio de Janeiro" "Salvador" "Santos" "Sao Jose dos Campos" "Sao Lourenco" "Sao Luis" "Sao Paulo" "Teresina"
"Uberlandia" "Vitoria")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Vector_USA = ciudades("Washington, DC" "Albany, New York" "Albuquerque, New Mexico" "Augusta, Maine" "Atlanta,
Georgia" "Baltimore, Maryland" "Billings, Montana" "Birmingham, Alabama" "Boise, Idaho" "Boston, Massachusetts"
"Buffalo, New York" "Charleston, South Carolina" "Cheyenne, Wyoming" "Chicago, Illinois" "Cheveland, Ohio"
"Columbia, South Calorina" "Columbus, Ohio" "Dallas/Ft Worth, Texas" "Denver, Colorado" "Des Moines, Iowa" "Detroit,
Michigan" "El Paso, Texas" "Fargo, North Dakota" "Grand Junction, Colorado" "Hartford, Connecticut" "Houston, Texas"
"Indianapolis, Indiana" "Jackson, Mississippi" "Jacksonville, Florida" "Kansas City, Kansas Missouri" "Las Vegas, Nevada"
"Little Rock, Arkansas" "Los Angeles, California" "Louisville, Kentucky" "Memphis, Tennessee" "Miami, Florida"
"Milwaukke, Wisconsin" "Minneapolis, Minnesota" "Nashville, Tennessee" "New Orleans, Louisiana" "New York City,
New York" "Norfolk, Virginia" "Oklahoma City, Oklahoma" "Omaha, Nebraska" "Orlando Florida" "Phidadelphia,
Pennsylvania" "Phoenix, Arizona" "Pittsburgh, Pennsylvania" "Reno, Nevada" "San Antonio, Texas" "San Diego,
California" "San Francisco, California")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N_ciudades
 N_numeros
%%%%%%%%
proc {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Col2}

                    Matriz_Nueva.Fila.Col =: Matriz_Origen.Fila2.Col2
end
```

```
proc {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion Fila Col Fila2 Cantidad_Aux Cont Aux}
 if Cont < Cantidad then
    if Posicion > Cantidad then
      if Aux < (Cantidad_Aux+1)  then
            {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Aux}
            {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux Cont
Aux+1}
            else
            {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux
Cantidad+1 Aux+1}
      end
   else
     {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Posicion}
     {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux-1 Cont Aux}
   end
  end
end

proc {Modificar_Matriz Matriz_Origen Matriz_Nueva Cantidad Posicion P}

   if Posicion > 1 then
     {For 1 Cantidad 1
      proc {$ Y}
            if Y == 1 then
              {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 Posicion Cantidad 1 1}
             % P=Posicion+1
            else
          if Y < (Cantidad-Posicion+2) then
                  {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 (Posicion+(Y-1)) Cantidad 1
1}
              else
                  {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 (Y-(Cantidad-Posicion+1))
Cantidad 1 1}

              end
            end
          end}
else
    {For 1 Cantidad 1
     proc {$ B}
      {For 1 Cantidad 1
            proc {$ A}
                  Matriz_Nueva.B.A =: Matriz_Origen.B.A
            end}
    end}
 end
end

proc {Guardar Contenido}
   try
      File={New Open.file init(name:"C:/TSP/"#Pais#"/"#Contenido.1.origen#".txt" flags:[write create truncate text binary])}
in
{File write(vs:"\n\r\n\r\n\r")}
   {File write(vs:"RESOLUCIÓN DEL PROBLEMA DEL VENDEDOR VIAJERO UTILIZANDO PROGRAMACIÓN
CON RESTRICCIONES")}
   {File write(vs:"\n\r\n\r\n\r")}
   {File write(vs:"JONATHAN ENRIQUE VIVANCO NAVARRO")}
   {File write(vs:"\n\r\n\r\n\r")}
   {File write(vs:"INFORME FINAL DEL PROYECTO PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO
DE EJECUCIÓN INFORMÁTICA")}
   {File write(vs:"\n\r\n\r\n\r")}
  {File write(vs:"Pais Seleccionado: "#Pais#" ")}
    {File write(vs:"\n\r\n\r")}
   {File write(vs:"Cantidad de Ciudades: "#Contenido.1.cantidad_ciudades)}
{File write(vs: "\n\r\n\rCiudad de Origen Seleccionada: "#Contenido.1.origen)}
{File write(vs: "\n\r\n\rCosto de Distancia Total: "#Contenido.1.costo_total)}
{File write(vs:" Kilometros")}
```

```
{File write(vs:"\n\r\n\rRuta: ")}
  {File write(vs:"\r\n\r\n")}

    if Pais == "Chile" then
          N_ciudades = 10
          N_numeros = 8
    end

    if Pais == "Brasil" then
           N_ciudades = 9
          N_numeros = 7
    end

    if Pais == "Estados Unidos" then
           N_ciudades = 5
          N_numeros = 7
    end

   {For 1 (Contenido.1.cantidad_ciudades+1) 1
    proc {$ I}
      if (I mod N_ciudades)== 0 then
           {File write(vs:"\r\n\r")}
      end

 {File write(vs:" "#Contenido.1.ruta.I)}
      if I < (Contenido.1.cantidad_ciudades+1) then
           {File write(vs:" - ")}
      end
      end}
{File write(vs:"\n\r\n\n\r\n\r\n\n\rDistancias: ")}
{File write(vs:"\r\n\r\n")}
    {For 1 (Contenido.1.cantidad_ciudades) 1
     proc {$ U}
       if (U mod N_numeros)== 0 then
           {File write(vs:"\n\r\n")}
       end
       {File write(vs:" "#Contenido.1.distancia.U)}
      {File write(vs:" Kilometros")}
      if U < (Contenido.1.cantidad_ciudades) then
           {File write(vs:" - ")}
      end
      end}
   {File write(vs:"\n\r\n\r")}
{File close}
catch _ then skip end

end

proc {Cambiar_orden_ciudades Orden_Original Orden_Nueva Cantidad Posicion}

  if Posicion > 1 then


    {For 1 Cantidad 1
         proc {$ Y}
                 if Y == 1 then

                   Orden_Nueva.1 = {String.toAtom Orden_Original.Posicion}
                 else

                   if Y < (Cantidad-Posicion+2) then
                      Orden_Nueva.Y = {String.toAtom Orden_Original.(Posicion+(Y-1))}
                   else
                     Orden_Nueva.Y = {String.toAtom Orden_Original.(Y-(Cantidad-Posicion+1))}
                   end
            end

             end}
```

```
else
      {For 1 Cantidad 1
       proc {$ I}
              Orden_Nueva.I = {String.toAtom Orden_Original.I}
       end}
   end
end

proc {TSP X}

Cantidad %%Cantidad de Ciudades de la matriz de distancias
   if Pais == "Chile" then
    Cantidad = {Width  Matriz.1}
   end
   if Pais == "Brasil" then
    Cantidad = {Width  MatrizB.1}
   end
   if Pais == "Estados Unidos" then
     Cantidad = {Width  MatrizUSA.1}
   end
   Ruta        = {FD.tuple camino Cantidad 1#Cantidad}
   Nombre_Ciudades = {MakeTuple ciudad (Cantidad+1)}
   Vector_Ciudades_modificada = {MakeTuple ciudad Cantidad}
   Matriz_Decision = {MakeTuple filas Cantidad}
    Matriz_Nueva = {MakeTuple filas Cantidad}
Rutas        = {FD.tuple camino (Cantidad+1) 1#Cantidad}
   Distancias    = {FD.tuple distancia Cantidad 1#9999}
    Suma_Costos     = {FD.decl} = {FD.sum Distancias '=:'}
{For 1 Cantidad 1
 proc {$ A}
    Matriz_Decision.A= {FD.tuple columnas Cantidad 0#1}
    Matriz_Nueva.A= {FD.tuple columnas Cantidad 0#9999}
 end}

in

   if Pais == "Chile" then
      {Modificar_Matriz Matriz Matriz_Nueva Cantidad Origen 1}
      {Cambiar_orden_ciudades Vector_Chile Vector_Ciudades_modificada Cantidad Origen}
   end
   if Pais == "Brasil" then
      {Modificar_Matriz MatrizB Matriz_Nueva Cantidad Origen 1}
      {Cambiar_orden_ciudades Vector_Brasil Vector_Ciudades_modificada Cantidad Origen}
   end
   if Pais == "Estados Unidos" then
      {Modificar_Matriz MatrizUSA Matriz_Nueva Cantidad Origen 1}
      {Cambiar_orden_ciudades Vector_USA Vector_Ciudades_modificada Cantidad Origen}
   end

%%%%%RESTRICCIONES%%%%%

%%% Para garantizar que no visite a la misma ciudad que  está ubicado

{FD.distinct Ruta}

  Ruta.Cantidad =: 1
   Rutas.1 =: 1

  {For 1 Cantidad 1
   proc {$ A}
     Rutas.(A+1) =: Ruta.A
    end}
%%%

X = resultado(cantidad_ciudades:Cantidad distancia:Distancias origen:Nombre_Ciudades.1 ruta:Nombre_Ciudades
costo_total: Suma_Costos)
```

```
if Pais == "Chile" then
    {FD.distribute generic(order:size  value:max) Ruta}
   end
   if Pais == "Brasil" then
    {FD.distribute generic(order:size  value:mid) Ruta}
   end
   if Pais == "Estados Unidos" then
    {FD.distribute generic(order:size  value:mid) Ruta}
   end

{For 1 Cantidad+1 1
  proc {$ A}
     Nombre_Ciudades.A = Vector_Ciudades_modificada.(Rutas.A)
  end}

   {For 1 Cantidad 1
  proc {$ A}
     Distancias.A = Matriz_Nueva.(Rutas.A).(Rutas.(A+1))
  end}
  end
proc {TSP_Orden Old New}
   Old.costo_total >: New.costo_total
end

%%devolver la mejor solución encontrada en cuestión de milisegundos MaxTime

proc {SearchBest ScriptP OrderP MaxTime ?Xs}

%%  el motor de búsqueda
        Engine={New Search.object script(ScriptP OrderP rcd:5)}

%%iterar a través de las soluciones, y devolver la mejor solución encontrada

      fun {Iterate CurrentSol}
            case {Engine next($)} of [X] then
               {Iterate [X]}
        else

                {Guardar CurrentSol}
               CurrentSol
            end
    end
in

   %% detener el motor después de MaxTime
      thread

            {Time.delay MaxTime} {Engine stop}

      end

      Xs={Iterate nil}

    end


    {Browse {SearchBest TSP TSP_Orden 60000}}
```

5. Código del Prototipo de Sistema en Oz utilizando QTk

```
%%% Prototipo de Sistema
%%%
%%% Autor:
%%%   Jonathan Vivanco Navarro Proyecto 2
%%%
%%% Resolución del Problema del Vendedor Viajero
%%%   Utilizando Programación con Restricciones

local
   [QTk]={Module.link ["x-oz://system/wp/QTk.ozf"]}
   Ventana  Texto1  Texto3  Texto2 AuxPanel Lista_aux  Lista_Valor
   Lista_Valor2 Opcion  Opcion_Origen  Lista_aux2
   Boton1 Boton2 Boton3 Boton4
Opcion_Origen   Lista_aux2   Boton1 Boton2 Boton3 Boton4    Cantidad
   Nombre_Ciudades  Vector_Ciudades_modificada  Matriz_Nueva  Contador
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%Chile%%%%%%%%%%%%%%%46 ciudades%%%%%%%%%%%%%%

   Lista_Chile = ["Angol" "Antofagasta" "Arica" "Calama" "Cartagena" "Castro" "Cauquenes" "Concepcion" "Coyhaique"
"Constitucion" "Copiapo" "Coquimbo" "Curico" "Chillan" "Illapel" "Iquique" "La Serena" "La Union" "Lebu" "Linares"
"Los Andes" "Los Angeles" "Los Lagos" "Osorno" "Ovalle" "Pucon" "Puerto Montt" "Puerto Varas" "La Ligua" "Punta
Arenas" "Rancagua" "San Antonio" "San Felipe" "San Fernando" "Santiago" "Talca" "Talcahuano" "Temuco" "Tocopilla"
"Valdivia" "Valparaiso" "Vallenar" "Villa Alemana" "Villarrica" "Vinia del Mar" "Zapallar"]
   Vector_Chile = ciudades("Angol" "Antofagasta" "Arica" "Calama" "Cartagena" "Castro" "Cauquenes" "Concepcion"
"Coyhaique" "Constitucion" "Copiapo" "Coquimbo" "Curico" "Chillan" "Illapel" "Iquique" "La Serena" "La Union" "Lebu"
"Linares" "Los Andes" "Los Angeles" "Los Lagos" "Osorno" "Ovalle" "Pucon" "Puerto Montt" "Puerto Varas" "La Ligua"
"Punta Arenas" "Rancagua" "San Antonio" "San Felipe" "San Fernando" "Santiago" "Talca" "Talcahuano" "Temuco"
"Tocopilla" "Valdivia" "Valparaiso" "Vallenar" "Villa Alemana" "Villarrica" "Vinia del Mar" "Zapallar")

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Matriz = distancia(


            angol(9999 1946 2638 2143 594 649 283 147 1241 374 1372 1027 378 166 849 2358 1038 351 161 270 646 60
288 378 972 237 480 464 719 2571 483 585 655 429 568 316 161 131 2129 297 674 1231 673 211 683 733)
            antofagasta(1946 9999 701 215 1391 2552 1730 1880 3011 1736 566 918 1571 1768 1184 492 887 2272 2020
1684 1337 1878 2210 2274 992 2158 2377 2385 1234 4451 1448 1397 1321 1516 1361 1618 1884 2038 185 2202 1313 722
1313 2132 1319 1250)
            arica(2638 701 9999 593 2084 3253 2423 2581 3712 2428 1261 1611 2263 2489 1876 316 1602 2965 2713 2376
2030 2579 2902 2975 1685 2850 3078 3078 1926 5152 2149 2030 2014 2209 2062 2319 2577 2739 536 2903 2020 1414
2006 2825 2011 1943)
            calama(594 215 593 9999 1589 2767 1927 2074 3360 1933 771 1116 1768 1978 1381 380 1106 2469 2217 1881
1534 2087 2407 2497 1189 2355 2598 2582 1431 4689 1657 1594 1518 1713 1574 1831 2081 2249 156 2415 1525 919
1510 2329 1516 1448)
            cartagena(584 1391 2084 1589 9999 1218 379 525 1811 384 818 472 219 429 295 1804 484 921 668 332 178 539
858 948 417 806 1050 1033 165 3140 136  9  177 165 112 282 533 701 1575 866 79  677 78 780 88 148)
            castro(649 2552 3253 2767 1218 9999 1218 801 414  999 1966 1651 1002 776 1474 3036 1664 314 778 894 1270
676 361 284 1596 506 175 190 1343 1994 1104 1209 1279 1054 1191 934 805 514 2753 385 1310 1855 1297 480 1308
1358)
            cauquenes(283 1730 2423 1927 379 1218 9999 140 1500 105 1157 811 162 118 634 2143 823 610 283 98 431 228
547 637 756 495 739 722 503 2829 268 369 439 214 353 100 147 390 1913 555 459 1015 457 469 468 518)
            concepcion(147 1880 2581 2074 525 801 140 9999 1260 245 1314 958 309 112 781 2372 993 493 143 201 577
127 430 523 903 378 626 606 650 2700 432 516 586 361 519 262 16 267 2060 452 639 1162 604 353 615 665)
            coyhaique(1241 3011 3712 3360 1811 414 1500 1260 9999 1591 2451 2244 1595 1384 2067 3503 2255 907 1370
1487 1863 1275 954 869 2189 1098 764 782 1936 1434 1701 1802 1872 1646 1786 1533 1397 1112 3346 973 1891 2448
1890 1073 1900 1950)
            constitucion(374 1736 2428 1933 384  999 105 245 1591 9999 1162 817 168 209 639 2148 828 701 388 112 436
319 638 728 762 587 830 814 509 2921 273 375 445 219 358 106 253 481 1919 647 464 1021 463 561 473 523)
            copiapo(1372 566 1261 771 818 1992 1157 1314 2451 1162 9999 345 997 1207 610 986 336 1699 1447 1110 764
1317 1636 1726 419 1584 1828 1812 660 3919 887 824 748 943 803 1060 1311 1479 757 1644 754 148 740 1559 745 677)
            coquimbo(1027 918 1611 1116 472 1651 811 958 2244 817 345 9999 652 862 265 1331  11  1353 1101 765 418
971 1291 1381 94 1239 1482 1466 315 3573 514 478 402 597 458 715 965 1133 1102 1299 409 204 394 1213 400 332)
```

```
curico(378 1571 2263 1768 219 1002 162 309 1595 168 997 652 9999 213 475 1983 664 704 443 116 271 322 642
732 597 590 833 817 344 2924 109 210 280 55 194 66 316 484 1754 650 300 856 298 564 309 359)
chillan(166 1768 2489 1978 429 776 118 112 1384 209 1207 862 213 9999 684 2193 873 493 241 105 481 111
430 521 807 379 622 606 554 2713 318 420 490 264 403 151 105 273 1964 439 509 1066 508 353 518 568)
illapel(849 1184 1876 1381 295 1474 634 781 2067 639 610 265 475 684 9999 1596 276 1176 924 587 241 794
1113 1204 184 1062 1305 1289 137 3396 364 301 225 420 280 538 788 956 1367 1122 232 469 217 1036 223 154)
iquique(2358 492 316 380 1804 3036 2143 2372 3503 2148 986 1331 1983 2193 1596 9999 1322 2685 2432 2096
1750 2303 2622 2712 1405 2570 2814 2798 1646 4905 1873 1809 1734 1929 1798 2046 2297 2465 229 2630 1740 1134
1726 2545 1731 1663)
la_serena(1038 887 1602 1106 484 1664 823 993 2255 828 336  11  664 873 276 1322 9999 1365 1113 776 430
983 1302 1393 85 1251 1494 1487 326 3585 553 490 414 609 469 726 977 1145 1092 1311 421 194 406 1225 412 343)
la_union(351 2272 2965 2469 921 314 610 493 907 701 1699 1353 704 493 1176 2685 1365 9999 480 597 973
385 64 44 1298 208 145 129 1045 2236 810 911 981 756 895 643 507 221 2456 66 1001 1558 999 182 1010 1060)
lebu(161 2020 2713 2217 668 778 283 143 1370 388 1447 1101 443 241 924 2432 1113 480 9999 344 720 216
417 507 1046 365 609 593 793 2700 558 659 729 504 643 390 155 260 2203 425 749 1305 747 340 758 808)
linares(270 1684 2376 1881 332 894 98 201 1487 112 1110 765 116 105 587 2096 776 597 344 9999 384 215 534
624 710 482 725 709 457 2816 221 323 393 167 306 54 209 377 1867 542 412 969 411 456 421 471)
los_andes(646 1337 2030 1534 178 1270 431 577 1863 436 764 418 271 481 241 1750 430 973 720 384 9999 591
910 1000 363 858 1102 1085 110 3192 161 188 20 217 78 334 585 753 1521 918 129 623 114 832 120 125)
los_angeles(60 1878 2579 2087 539 676  228 127 1275 319 1317 971 322 111 794 2303 983 385 216 215 591
9999 322 412 916 270 513 497 663 2604 428 529 600 374 513 261 153 165 2074 330 619 1176 618 244 628 678)
los_lagos(288 2210 2902 2407 858 361 547 430 954 638 1636 1291 642 430 1113 2622 1302 64 417 534 910 322
9999 91 1236 145 193 176 983 2283 747 849 919 693 832 580 444 158 2393 67 938 1495 937 119 947 997)
osorno(378 2274 2975 2497 948 284 637 523 869 728 1726 1381 732 521 1204 2712 1393 44 507 624 1000 412
91 9999 1326 235 107 91 1073 2199 838 939 1009 784 923 670 534 249 2483 110 1029 1585 1057 210 1038 1088)
ovalle(972 992 1685 1189 417 1596 756 903 2189 762 419 94 597 807 184 1405 85 1298 1046 710 363 916 1236
1326 9999 1184 1427 1411 260 3518 486 423 348 542 403 660 910 1078 1176 1244 354 278 339 1158 345 277)
pucon(237 2158 2850 2355 806 506 495 378 1098 587 1584 1239 590 379 1062 2570 1251 208 365 482 858 270
145 235 1184 9999 337 321 931 2428 696 797 867 642 781 528 392 107 2341 153 887 1443 885 26 896 946)
puerto_montt(480 2377 3078 2598 1050 175 739 626 764 830 1828 1482 833 622 1305 2814 1494 145 609 725
1102 513 193 107 1427 337 9999 21 1174 2300 939 1040 1110 885 1024 771 636 350 2585 212 1130 1686 1128 311 1139
1189)
puerto_varas(464 2385 3078 2582 1033 190 722 606 782 814 1812 1466 817 606 1289 2798 1487 129 593 709
1085 497 176 91 1411 321 21 9999 1158 2284 923 1024 1094 869 1008 755 620 334 2568 196 1114 1670 1112 295 1123
1173)
la_ligua(719 1234 1926 1431 165 1343 503 650 1936 509 660 315 344 554 137 1646 326 1045 793 457 110 663
983 1073 260 931 1174 1158 9999 3265 233 170 94 289 150 407 657 825 1417 991 101 519 86 905 92 34)
punta_arenas(2571 4451 5152 4689 3140 1994 2829 2700 1434 2921 3919 3573 2924 2713 3396 4905 3585 2236
2700 2816 3192 2604 2283 2199 3518 2428 2300 2284 3265 9999 3030 3131 3201 2976 3115 2862 2727 2441 4675 2303
3221 3777 3219 2402 3230 3280)
rancagua(483 1448 2149 1657 136 1104 268 432 1701 273 887 514 109 318 364 1873 553 810 558 221 161 428
747 838 486 696 939 923 233 3030 9999 136 169 54 83 172 422 590 1644 756 189 746 188 670 198 248)
san_antonio(585 1397 2030 1594  9  1209 369 516 1802 375 824 478 210 420 301 1809 490 911 659 323 188 529
849 939 423 797 1040 1024 170 3131 136 9999 197 155 112 273 523 691 1580 857 85 682 83 771 94 154)
san_felipe(655 1321 2014 1518 177 1279 439 586 1872 445 748 402 280 490 225 1734 414 981 729 393 20 600
919 1009 347 867 1110 1094 94 3201 169 197 9999 225 87 343 593 761 1505 927 113 607 98 841 104 109)
san_fernando(429 1516 2209 1713 165 1054 214 361 1646 219 943 597 55 264 420 1929 609 756 504 167 217
374 693 784 542 642 885 869 289 2976 54 155 225 9999 139 117 368 536 1700 702 245 801 243 616 254 304)
santiago(568 1377 2070 1574 112 1191 353 519 1786 358 803 458 194 403 280 1798 469 895 643 306 78 513 832
923 403 781 1024 1008 150 3115 83 112 87 139 9999 257 507 675 1560 841 116 662 107 755 125 164)
talca(316 1618 2319 1831 282 934 100 262 1533 106 1060 715 66 151 538 2046 726 643 390 54 334 261 580 670
660 528 771 755 407 2862 172 273 343 117 257 9999 255 423 1817 588 363 919 361 502 372 422)
talcahuano(161 1884 2577 2081 533 805 147 16 1397 253 1311 965 316 105 788 2297 977 507 155 209 585 153
444 534 910 392 636 620 657 2727 422 523 593 368 507 255 9999 287 2068 452 613 1170 611 367 622 672)
temuco(131 2038 2739 2249 701 514 390 267 1112 481 1479 1133 484 273 956 2465 1145 221 260 377 753 165
158 249 1078 107 350 334 825 2441 590 691 761 536 675 423 287 9999 2236 167 781 1338 779 81 790 840)
tocopilla(2129 185 536 156 1575 2753 1913 2060 3346 1919 757 1102 1754 1964 1367 229 1092 2456 2203 1867
1521 2074 2393 2483 1176 2341 2585 2568 1417 4675 1644 1580 1505 1700 1560 1817 2068 2236 9999 2401 1511 905
1496 2315 1502 1434)
valdivia(297 2202 2903 2415 866 385 555 452 973 647 1644 1299 650 439 1122 2630 1311 66 425 542 918 330
67 110 1244 153 212 196 991 2303 756 857 927 702 841 588 452 167 2401 9999 947 1503 945 128 956 1006)
valparaiso(674 1319 2020 1525  79 1310 459 639 1891 464 754 409 300 509 232 1740 421 1001 749 412 129 619
938 1029 354 887 1130 1114 101 3221 189 85 113 245 116 363 613 781 1511 947 9999 613 25 861  8  72)
vallenar(1231 722 1414 919 677 1855 1015 1162 2448 1021 148 204 856 1066 469 1134 194 1558 1305 969 623
1176 1495 1585 278 1443 1686 1670 519 3777 746 682 607 801 662 919 1170 1338 905 1503 613 9999 598 1417 604 536)
villa_alemana(673 1313 2006 1510 78 1297 457 604 1890 463 740 394 298 508 217 1726 406 999 747 411 114
618 937 1057 339 885 1128 1112 86 3219 188 83 98 243 107 361 611 779 1496 945 25 598 9999 859 20 80)
villarrica(211 2132 2825 2329 780 480 469 353 1073 561 1559 1213 564 353 1036 2545 1225 182 340 456 832
244 119 210 1158 26 311 295 905 2402 670 771 841 616 755 502 367 81 2315 128 861 1417 859 9999 870 920)
```

```
            vina_del_mar(683 1319 2011 1516 88 1308 468 615 1900 473 745 400 309 518 223 1731 412 1010 758 421 120
628 947 1038 345 896 1139 1123 92 3230 198 94 104 254 125 372 622 790 1502 956  8  604 20 870 9999 63)
            zapallar(733 1250 1943 1448 148 1358 518 665 1950 523 677 332 359 568 154 1663 343 1060 808 471 125 678
997 1088 277 946 1189 1173 34 3280 248 154 109 304 164 422 672 840 1434 1006 72 536 80 920 63 9999)
)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%Brasil%%%%%%%%%%%%%%36 ciudades%%%%%%%%%%%%%%
Lista_Brasil = ["Brasilia" "Aracaju" "Belem" "Belo Horizonte" "Boa Vista" "Campinas" "Campo Grande" "Caxias Do Sul"
"Cuiaba" "Curitiba" "Feria de Santana" "Florianopolis" "Fortaleza" "Goiania" "Joao Pessoa" "Joinville" "Juiz de Fora"
"Londrina" "Maceio" "Manaus" "Natal" "Pelotas" "Porto Alegre" "Porto Velho" "Recife" "Ribeirao Preto" "Rio Branco"
"Rio de Janeiro" "Salvador" "Santos" "Sao Jose dos Campos" "Sao Lourenco" "Sao Luis" "Sao Paulo" "Teresina"
"Uberlandia" "Vitoria"]

Vector_Brasil = ciudad("Brasilia" "Aracaju" "Belem" "Belo Horizonte" "Boa Vista" "Campinas" "Campo Grande" "Caxias
Do Sul" "Cuiaba" "Curitiba" "Feria de Santana" "Florianopolis" "Fortaleza" "Goiania" "Joao Pessoa" "Joinville" "Juiz de
Fora" "Londrina" "Maceio" "Manaus" "Natal" "Pelotas" "Porto Alegre" "Porto Velho" "Recife" "Ribeirao Preto" "Rio
Branco" "Rio de Janeiro" "Salvador" "Santos" "Sao Jose dos Campos" "Sao Lourenco" "Sao Luis" "Sao Paulo" "Teresina"
"Uberlandia" "Vitoria")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MatrizBrasil =distancias2(

brasilia(9999 1652 2120 716 4275 921 1134 1900 1133 1366 1415 1673 2378 209 2245 1503 992 1083 1928 3490 2422
2290 2027 2589 2220 706 3123 1148 1446 1087 1087 2157 1015 1789 435 1238)
aracaju(1652 9999 2079 1578 6000 2182 541 3169 2773 2595 322 2892 1183 1849 611 2722 1718 2584 294 5215 788 3559
3296 4229 501 2106 4763 1855 356 2249 2086 1578 2188 1142 2137 1408)
belem(2120 2079 9999 2824 6083 2842 2942 3727 2941 3193 1984 3500 1611 2017 2161 3330 3100 2891 2173 5298 2108
4117 3854 4397 2074 2622 4931 3250 2100 3005 3008 806 2933 947 2367 3108)
belo_horizonte(716 1578 2824 9999 4736 601 1453 1585 1594 1004 1256 1301 2528 906 2171 1131 272 1003 1854 3951
2348 1975 1712 3050 2061 523 3584 434 1372 658 611 2738 586 2302 556 524)
boa_vista(4275 6000 6083 4736 9999 4665 3836 5355 3142 4821 5378 5128 6548 4076 6539 4958 4978 4445 6276 785
6770 5611 5348 1686 6483 4445 2230 5159 5749 4828 4833 6120 4756 6052 4190 5261)
campinas(921 2182 2842 601 4665 9999 1012 1057 1523 476 1866 773 3133 835 2775 600 541 526 2458 3880 2952 1440
1177 2979 2665 238 3513 511 1982 171 174 2879   99 2698 497 959)
campo_grande(1134 541 2942 1453 3836 1012 9999 1525 694 991 2537 1298 3407 935 3357 1128 1512 615 3040 3051
3537 1781 1518 2150 3247 930 2684 1444 2568 1086 1107 2979 1014 2911 894 1892)
caxias_do_sul(1900 3169 3727 1585 5355 1057 1525 9999 2213 584 2847 478 4115 1720 3762 606 1498 913 3445 4570
3939 394 150 3669 3652 1215 4203 1426 2963 1054 1089 3764 982 3677 1476 1874)
cuiaba(1133 2773 2941 1594 3142 1523 694 2213 9999 1679 2539 1986 3406 934 3366 1816 1836 1303 3049 2357 3543
2469 2206 1456 3256 1303 1990 2017 2567 1686 1689 2978 1614 2910 1048 2119)
feria_de_santana(1366 2595 3193 1004 4821 476 991 584 1679 9999 2269 300 3541 1186 3188 130 914 379 2871 4036
3365 974 711 3135 3078 681 3669 852 2385 480 515 3230 408 3143 942 1300)
florianopolis(1415 322 1984 1256 5378 1866 2537 2847 2539 2269 9999 2566 1273 1528 915 2396 1396 2262 598 4893
1092 3237 2974 3907 805 1784 4441 1533 116 1918 1764 1483 1846 1047 1816 1124)
fortaleza(1673 2892 3500 1301 5128 773 1298 478 1986 300 2566 9999 3838 1493 3485 180 1221 686 3168 4343 3662 739
476 3442 3375 988 3976 1144 2682 777 807 3537 705 3450 1249 1597)
goiania(2378 1183 1611 2528 6548 3133 3407 4115 3406 3541 1273 3838 9999 2482 688 3668 2668 3356 1075 5763 537
4505 4242 4865 800 2978 5396 2805 1389 3199 3036 1070 3127 634 2707 2397)
joao_pessoa(209 1849 2017 906 4076 835 935 1720 934 1186 1528 1493 2482 9999 2442 1323 1174 884 2105 3291 2619
2110 1847 2390 2332 615 2924 1338 1643 998 1005 2054 926 1986 360 1428)
joinville(2245 611 2161 2171 6539 2775 3357 3762 3366 3188 915 3485 688 2442 9999 3315 2311 3177 395 5808 185
4152 3710 4907 120 2699 5356 2448 949 2842 2679 1660 2770 1224 2730 2001)
juiz_de_fora(1503 2722 3330 1131 4958 600 1128 606 1816 130 2396 180 3668 1323 3315 9999 1046 516 2998 4173 3492
903 640 3272 3205 818 3806 974 2512 607 2679 3367 535 3280 1079 1427)
londrina(992 1718 3100 272 4978 541 1512 1498 1836 914 1396 1221 2668 1174 2311 1046 9999 1024 1994 4193 2488
1888 1625 3292 2201 676 3826 184 1512 578 415 2874 506 2438 798 519)
maceio(1083 2584 2891 1003 4445 526 615 913 1303 379 2262 686 3356 884 3177 516 1024 9999 2860 3660 3354 1303
1040 1958 3067 478 3293 953 2374 603 621 2928 538 2860 716 1406)
manaus(1928 294 2173 1854 6276 2458 3040 3445 3049 2871 598 3168 1075 2105 395 2998 1994 2860 9999 5491 572
3835 3572 4505 285 2382 5124 2131 632 2525 2362 1672 2453 1236 2413 1684)
natal(3490 5215 5298 3951 785 3880 3051 4570 2357 4036 4893 4343 5763 3291 5808 4173 4193 3660 5491 9999 5985
4826 4563 901 5698 3660 1445 4374 5009 4043 4046 5335 3971 5267 3405 4476)
pelotas(2422 788 2108 2348 6770 2952 3537 3939 3543 3365 1092 3662 537 2619 185 3492 2488 3354 572 5985 9999
4329 4066 4999 297 2876 5533 2625 1126 3019 2859 1607 2947 1171 2907 2178)
porto_alegre(2290 3559 4117 1975 5611 1440 1781 394 2469 974 3237 739 4505 2110 4152 903 1888 1303 3835 4826
4329 9999 271 3925 4042 1605 4459 1816 3353 1444 1479 4154 1372 4067 1866 2264)
porto_velho(2027 3296 3854 1712 5348 1177 1518 150 2206 711 2974 476 4242 1847 3710 640 1625 1040 3572 4563
4066 271 9999 3662 3779 1342 4196 1553 3127 1181 1216 3891 1109 3804 1603 2001)
```

```
recife(2589 4229 4397 3050 1686 2979 2150 3669 1456 3135 3907 3442 4865 2390 4907 3272 3292 1958 4505 901 4999
3925 3662 9999 4712 2759 544 3473 4023 3142 3145 4434 3070 4366 2504 3575)
ribeirao_preto(2220 501 2074 2061 6483 2665 3247 3652 3256 3078 805 3375 800 2332 120 3205 2201 3067 285 5698 297
4042 3779 4712 9999 2589 5243 2338 839 2732 2569 1573 2660 1137 2620 1891)
rio_branco(706 2106 2622 523 4445 238 930 1215 1303 681 1784 988 2978 615 2699 818 676 478 2382 3660 2876 1605
1342 2759 2589 9999 3293 725 1900 391 408 2659 319 2483 281 1048)
rio_de_janeiro(3123 4763 4931 3584 2230 3513 2684 4203 1990 3669 4441 3976 5396 2924 5356 3806 3826 3293 5124
1445 5533 4459 4196 544 5243 3293 9999 4007 4457 3676 3679 4968 3604 4900 3038 4109)
salvador(1148 1855 3250 434 5159 511 1444 1426 2017 852 1533 1144 2805 1338 2448 974 184 953 2131 4374 2625 1816
1553 3473 2338 725 4007 9999 1649 501 343 3015 429 2579 979 521)
santos(1446 356 2100 1372 5749 1982 2568 2963 2567 2385 116 2682 1389 1643 949 2512 1512 2374 632 5009 1126 3353
3127 4023 839 1900 4457 1649 9999 2034 1880 1599 1962 1163 1932 1202)
sao_jose_dos_campos(1087 2249 3005 658 4828 171 1086 1054 1686 480 1918 777 3199 998 2842 607 578 603 2525 4043
3019 1444 1181 3142 2732 391 3676 501 2034 9999 169 3042 72  2864 662 954)
sao_lourenço(1087 2086 3008 611 4833 174 1107 1089 1689 515 1764 807 3036 1005 2679 2679 415 621 2362 4046 2859
1479 1216 3145 2569 408 3679 343 1880 169 9999 3049 97  2810 669 791)
sao_luis(2157 1578 806 2738 6120 2879 2979 3764 2978 3230 1483 3537 1070 2054 1660 3367 2874 2928 1672 5335 1607
4154 3891 4434 1573 2659 4968 3015 1599 3042 3049 9999 2970 446 2404 2607)
sao_paulo(1015 2188 2933 586 4756   99 1014 982 1614 408 1846 705 3127 926 2770 535 506 538 2453 3971 2947 1372
1109 3070 2660 319 3604 429 1962 72  97  2970 9999 2792 590 882)
teresina(1789 1142 947 2302 6052 2698 2911 3677 2910 3143 1047 3450 634 1986 1224 3280 2438 2860 1236 5267 1171
4067 3804 4366 1137 2483 4900 2579 1163 2864 2810 446 2792 9999 2212 2171)
uberlandia(435 2137 2367 556 4190 497 894 1476 1048 942 1816 1249 2707 360 2730 1079 798 716 2413 3405 2907 1866
1603 2504 2620 281 3038 979 1932 662 669 2404 590 2212 9999 1081)
vitoria(1238 1408 3108 524 5261 959 1892 1874 2119 1300 1124 1597 2397 1428 2001 1427 519 1406 1684 4476 2178
2264 2001 3575 1891 1048 4109 521 1202 954 791 2607 882 2171 1081 9999)
)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%Estados Unidos%%%%%%%%%52 ciudades%%%%%%%%%%

   Lista_USA = ["Washington, DC" "Albany, New York" "Albuquerque, New Mexico" "Augusta, Maine" "Atlanta, Georgia"
"Baltimore, Maryland" "Billings, Montana" "Birmingham, Alabama" "Boise, Idaho" "Boston, Massachusetts" "Buffalo,
New York" "Charleston, South Carolina" "Cheyenne, Wyoming" "Chicago, Illinois" "Cheveland, Ohio" "Columbia, South
Calorina" "Columbus, Ohio" "Dallas/Ft Worth, Texas" "Denver, Colorado" "Des Moines, Iowa" "Detroit, Michigan" "El
Paso, Texas" "Fargo, North Dakota" "Grand Junction, Colorado" "Hartford, Connecticut" "Houston, Texas" "Indianapolis,
Indiana" "Jackson, Mississippi" "Jacksonville, Florida" "Kansas City, Kansas Missouri" "Las Vegas, Nevada" "Little Rock,
Arkansas" "Los Angeles, California" "Louisville, Kentucky" "Memphis, Tennessee" "Miami, Florida" "Milwaukke,
Wisconsin" "Minneapolis, Minnesota" "Nashville, Tennessee" "New Orleans, Louisiana" "New York City, New York"
"Norfolk, Virginia" "Oklahoma City, Oklahoma" "Omaha, Nebraska" "Orlando Florida" "Phidadelphia, Pennsylvania"
"Phoenix, Arizona" "Pittsburgh, Pennsylvania" "Reno, Nevada" "San Antonio, Texas" "San Diego, California" "San
Francisco, California"]

Vector_USA = ciudades("Washington, DC" "Albany, New York" "Albuquerque, New Mexico" "Augusta, Maine" "Atlanta,
Georgia" "Baltimore, Maryland" "Billings, Montana" "Birmingham, Alabama" "Boise, Idaho" "Boston, Massachusetts"
"Buffalo, New York" "Charleston, South Carolina" "Cheyenne, Wyoming" "Chicago, Illinois" "Cheveland, Ohio"
"Columbia, South Calorina" "Columbus, Ohio" "Dallas/Ft Worth, Texas" "Denver, Colorado" "Des Moines, Iowa" "Detroit,
Michigan" "El Paso, Texas" "Fargo, North Dakota" "Grand Junction, Colorado" "Hartford, Connecticut" "Houston, Texas"
"Indianapolis, Indiana" "Jackson, Mississippi" "Jacksonville, Florida" "Kansas City, Kansas Missouri" "Las Vegas, Nevada"
"Little Rock, Arkansas" "Los Angeles, California" "Louisville, Kentucky" "Memphis, Tennessee" "Miami, Florida"
"Milwaukke, Wisconsin" "Minneapolis, Minnesota" "Nashville, Tennessee" "New Orleans, Louisiana" "New York City,
New York" "Norfolk, Virginia" "Oklahoma City, Oklahoma" "Omaha, Nebraska" "Orlando Florida" "Phidadelphia,
Pennsylvania" "Phoenix, Arizona" "Pittsburgh, Pennsylvania" "Reno, Nevada" "San Antonio, Texas" "San Diego,
California" "San Francisco, California")
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MatrizUSA = distancias_usa(

washington__DC(9999 645 1000 2984 1161 64  3403 1193 3822 725 661 854 2645 1145 580 774 693 2113 2613 1709 838
3113 2145 3032 580 2209 935 1564 1161 1677 3903 709 4274 967 1371 1709 1306 1758 1064 1774 387 306 2145 1838
1371 209 3709 403 4145 2564 4193 4580)
albany__New_York(645 9999 1629 3290 709 548 3387 1725 4064 274 483 1419 2887 1322 774 1322 1032 2709 2951
1903 1048 3580 2403 3387 177 2855 1306 2177 1725 2096 4242 2193 4597 1387 1983 2258 1500 2016 1661 2322 258 822
2419 2113 1983 419 4016 725 4451 3177 4597 4806)
albuquerque__New_Mexico(1000 1629 9999 2258 2145 1048 2903 241 3580 1790 1467 467 2387 1145 1177 338 951 1322
2306 1548 1177 2322 2242 2677 1548 1274 854 645 500 1322 3193 887 3532 677 612 1064 1290 1806 403 774 1371 903
1338 1629 693 1209 2951 1096 3887 1613 3467 4000)
augusta__Maine(2984 3290 2258 9999 3967 3048 1596 2016 1516 3580 2855 2742 871 2113 2564 2613 2354 1032 709
1564 2516 435 2080 693 3387 1371 2048 1693 2645 1258 851 1419 1306 2080 1629 3177 2225 1967 1983 1935 3226 3048
871 1435 2806 3145 741 2629 1645 1177 1306 1790)
```

atlanta__Georgia(1161 709 2145 3967 9999 1080 3984 2435 4774 435 1177 1967 3580 2016 1403 1951 1629 3258 3613 2532 1709 4242 3048 4048 596 3419 1967 2822 2371 2774 4887 2855 5322 2016 2629 2951 2129 2645 2290 2919 822 1435 3113 3758 2613 951 4709 1387 5064 3693 5274 5451)
baltimore__Maryland(64 548 1048 3048 1080 9999 3096 1242 3887 693 596 919 2693 1161 580 838 693 2193 2645 1677 822 3193 2177 3064 516 2258 919 1629 1209 1725 3871 1677 4274 1032 1467 1758 1274 1790 1129 1838 322 370 3113 1854 1435 161 3726 403 4226 2629 4322 4548)
billings__Montana(3403 3387 2903 1596 3984 3096 9999 2871 983 3548 2774 3516 741 1983 2596 3322 2596 2290 887 1580 2451 2032 983 1080 3387 2564 2274 2774 3548 1677 1693 2290 2000 2467 2345 4064 1871 1338 2580 2951 3290 3355 1854 1467 3596 3193 1935 2677 1548 2451 2113 1919)
birmingham__Alabama(1193 1725 241 2016 2435 1242 2871 9999 3338 1983 1451 709 2290 1064 1177 580 919 1032 2145 1338 1225 2048 2096 2629 1742 1129 774 403 741 1177 2951 693 3306 580 419 1209 1193 1725 306 564 1580 1145 1161 1419 887 1403 2709 1274 3677 1387 3209 3822)
boise__Idaho(3822 4064 3580 1516 4774 3887 983 3338 9999 4338 3564 4016 1177 2758 3274 3855 3177 2564 1354 2193 3193 1935 1951 1016 4209 2935 2984 3290 3984 2290 1080 2855 1371 3113 2935 4629 2806 2387 3193 3467 4016 4064 2322 2000 4258 3919 1580 3435 693 2693 1580 1048)
boston__Massachusetts(725 274 1790 3580 435 693 3548 1983 4338 9999 758 1516 3096 1613 1064 1548 1290 2822 3226 2161 1290 3838 2661 3661 177 2951 1500 2354 1871 2322 4435 2322 4871 1548 2161 2451 1758 2242 1758 2435 338 935 2725 2371 2096 516 4306 919 4629 3258 4645 5048)
buffalo__New_York(661 483 1467 2855 1177 596 2774 1451 3564 758 9999 1500 2419 871 306 1290 532 2193 2500 1371 580 3177 1919 2887 677 2403 822 1806 1758 1629 3677 1709 4161 887 1483 2258 1016 1532 1161 2032 596 967 1967 1645 1935 580 3580 354 3951 2629 4080 4306)
charleston__South_Carolina(854 1419 467 2742 1967 919 3516 709 4016 1516 1500 9999 2758 1467 1177 177 1048 1758 2774 1935 1354 2742 2435 3161 1338 1693 1161 1080 387 1790 3677 1338 4016 938 1129 951 1645 2145 871 1177 1242 709 1838 2048 612 1064 3419 1048 4387 2064 3903 4548)
cheyenne__Wyoming(2645 2887 2387 871 3580 2693 741 2290 1177 3096 2419 2758 9999 1580 2145 2677 2096 1419 161 1032 2000 1274 1258 564 2984 1790 1742 2048 2822 1096 1371 1725 1806 1935 1838 3467 1613 1419 2032 2225 2822 2903 1112 806 3016 2838 1516 2306 1564 1709 1919 1919)
chicago__Illinois(1145 1322 1145 2113 2016 1161 1983 1064 2758 1613 871 1467 1580 9999 564 1290 580 1483 1645 548 451 2354 1048 2048 1467 1758 306 1209 1629 871 2871 1064 3306 483 871 2258 145 661 758 1483 1306 1403 1338 774 1854 1274 2806 774 3145 1951 3371 3500)
cheveland__Ohio(580 774 1177 2564 1403 580 2596 1177 3274 1064 306 1177 2145 564 9999 983 225 1919 2193 1064 274 2806 1613 2596 903 2113 516 1516 1467 1322 3371 1403 3838 564 1177 2016 709 1225 854 1709 758 822 1661 1338 1693 693 3274 209 3645 2338 3855 4000)
columbia__South_Calorina(774 1322 338 2613 1951 838 3322 580 3855 1548 1290 177 2677 1290 983 9999 854 1661 2613 1790 1161 2629 2306 2967 1306 1661 983 967 483 1645 3532 1225 3903 806 1000 1032 1419 1951 709 1112 1161 629 1709 1903 709 983 3274 951 4209 1983 3838 4468)
columbus__Ohio(693 1032 951 2354 1629 693 2596 919 3177 1290 532 1048 2096 580 225 854 9999 1693 2000 1064 306 2580 1532 2403 1064 1887 290 1290 1338 1096 3258 1177 3613 338 951 1887 758 1258 612 1483 903 903 1483 1274 1548 774 3064 306 3564 2113 3645 3919)
dallas_Ft_Worth__Texas(2113 2709 1322 1032 3258 2193 2290 1032 2564 2822 2193 1758 1419 1483 1919 1661 1693 9999 1258 1129 1871 1000 1758 1580 2725 403 1435 661 1677 822 1983 516 2258 1338 725 2161 1645 1532 1064 838 2516 2177 338 1064 1774 2322 1613 1951 2693 435 2177 2822)
denver__Colorado(2613 2951 2306 709 3613 2645 887 2145 1354 3226 2500 2774 161 1645 2193 2613 2000 1258 9999 1080 2064 1112 1419 403 3226 1661 1709 1935 2806 983 1225 1516 1661 1806 1677 3403 1677 1483 1903 2064 2887 2855 1016 871 3032 2806 1306 2306 1661 1532 1774 2032)
des_Moines__Iowa(1709 1903 1548 1564 2532 1677 1580 1338 2193 2161 1371 1935 1032 548 1064 1790 1064 1129 1080 9999 951 1822 774 1483 2016 1516 774 1338 2000 322 2306 903 2758 951 1000 2564 596 403 1112 1580 1806 1854 887 209 2193 1758 2306 1258 2580 1580 2855 2951)
detroit__Michigan(838 1048 1177 2516 1709 822 2451 1225 3193 1290 580 1354 2000 451 274 1161 306 1871 2064 951 9999 2693 1483 2451 1177 2064 451 1548 1693 1242 3258 1419 3693 612 1161 2242 580 1112 871 1725 1048 1177 1661 1177 1887 983 3242 483 3548 2338 3822 3871)
el_Paso__Texas(3113 3580 2322 435 4242 3193 2032 2048 1935 3838 3177 2742 1274 2354 2806 2629 2580 1000 1112 1822 2693 9999 2338 1080 3693 1193 2290 1661 2548 1516 1161 1516 1322 2338 1725 3064 2484 2354 2064 1774 3467 3145 1096 2016 2709 3338 709 2871 1887 903 1177 1935)
fargo__North_Dakota(2145 2403 2242 2080 3048 2177 983 2096 1951 2661 1919 2435 1258 1048 1613 2306 1532 1758 1419 774 1483 2338 9999 1790 2516 2096 1338 2113 2725 1000 2484 1677 2919 1516 1822 3226 919 387 1790 2225 2338 2371 1403 693 2919 2306 2725 1822 2548 2145 3016 2935)
grand_Junction__Colorado(3032 3387 2677 693 4048 3064 1080 2629 1016 3661 2887 3161 564 2048 2596 2967 2403 1580 403 1483 2451 1080 1790 9999 3467 1967 2145 2274 3161 1387 822 1854 1258 2209 2048 3709 2080 1871 2290 2371 3290 3258 1322 1274 3338 3209 935 2725 1209 1758 1371 1580)
hartford__Connecticut(580 177 1548 3387 596 516 3387 1742 4209 177 677 1338 2984 1467 903 1306 1064 2725 3226 2016 1177 3693 2516 3467 9999 2790 1354 2177 1693 2161 4290 2161 4677 1403 1935 2290 1613 2129 1613 2306 193 790 2645 2209 1774 338 4145 758 4516 3080 4677 4968)
houston__Texas(2209 2855 1274 1371 3419 2258 2564 1129 2935 2951 2403 1693 1790 1758 2113 1661 1887 403 1661 1516 2064 1193 2096 1967 2790 9999 1613 677 1435 1193 2371 693 2484 1516 919 1919 1903 1903 1258 564 2596 2177 741 1387 1580 2435 1871 2209 3032 322 2403 3080)

```
indianapolis__Indiana(935 1306 854 2048 1967 919 2274 774 2984 1500 822 1161 1742 306 516 983 290 1435 1709 774
451 2290 1338 2145 1354 1613 9999 1080 1354 806 2967 983 3322 177 758 1919 435 951 451 1290 1177 1145 1193 983
1564 1064 2790 580 3322 1919 3355 3693)
jackson__Mississippi(1564 2177 645 1693 2822 1629 2774 403 3290 2354 1806 1080 2048 1209 1516 967 1290 661 1935
1338 1548 1661 2113 2274 2177 677 1080 9999 983 1112 2629 419 2935 951 338 1467 1338 1709 661 338 1967 1532 903
1403 1129 1790 2354 1516 3355 1080 2838 3467)
jacksonville__Florida(1161 1725 500 2645 2371 1209 3548 741 3984 1871 1758 387 2822 1629 1467 483 1338 1677 2806
2000 1693 2548 2725 3161 1693 1435 1354 983 9999 1838 3613 1322 3790 1177 1112 564 1790 2354 903 903 1532 1000
1854 2129 225 1371 3226 1338 1338 1742 3806 4435)
kansas_City__Kansas_Missouri(1677 2096 1322 1258 2774 1725 1677 1177 2290 2322 1629 1790 1096 871 1322 1645
1096 822 983 322 1242 1516 1000 1387 2161 1193 806 1112 1838 9999 2209 677 2548 822 774 2387 903 709 935 1354
1983 1871 564 306 2016 1887 2000 1403 2645 1258 2564 3000)
las_Vegas__Nevada(3903 4242 3193 851 4887 3871 1693 2951 1080 4435 3677 3677 1371 2871 3371 3532 3258 1983
1225 2306 3258 1161 2484 822 4290 2371 2967 2629 3613 2209 9999 2354 435 3016 2580 4145 2903 2677 2919 2790
4145 4000 1790 2096 3790 4000 467 3564 725 2080 548 919)
little_Rock__Arkansas(709 2193 887 1419 2855 1677 2290 693 2855 2322 1709 1338 1725 1064 1403 1225 1177 516 1516
903 1419 1516 1677 1854 2161 693 983 419 1322 677 2354 9999 2725 822 225 1871 1242 1338 564 677 2016 1645 548
935 1548 1838 2145 1451 645 2693 3209 3629)
los_Angeles__California(4274 4597 3532 1306 5322 4274 2000 3306 1371 4871 4161 4016 1806 3306 3838 3903 3613
2258 1661 2758 3693 1322 2919 1258 4677 2484 3322 2935 3790 2548 435 2725 9999 3516 2919 4387 3338 3000 3242
3000 4500 4338 2161 2532 3919 4355 629 3919 758 2242 209 629)
louisville__Kentucky(967 1387 677 2080 2016 1032 2467 580 3113 1548 887 938 1935 483 564 806 338 1338 1806 951
612 2338 1516 2209 1403 1516 177 951 1177 822 3016 822 3516 9999 612 1758 612 1145 274 1129 1242 1048 1209 1129
1387 1145 2822 629 3451 1774 3371 3790)
memphis__Tennessee(1371 1983 612 1629 2629 1467 2345 419 2935 2161 1483 1129 1838 871 1177 1000 951 725 1677
1000 1161 1725 1822 2048 1935 919 758 338 1112 774 2580 225 2919 612 9999 1613 1000 1467 338 661 1774 1419 774
1032 1258 1629 2371 1225 3274 1177 2919 3419)
miami__Florida(1709 2258 1064 3177 2951 1758 4064 1209 4629 2451 2258 951 3467 2258 2016 1032 1887 2161 3403
2564 2242 3064 3226 3709 2290 1919 1919 1467 564 2387 4145 1871 4387 1758 1613 9999 2354 2855 1467 1387 2145
1564 2419 2645 370 1983 3790 1903 4829 2242 4322 4984)
milwaukke___Wisconsin(1306 1500 1290 2225 2129 1274 1871 1193 2806 1758 1016 1645 1613 145 709 1419 758 1645
1677 596 580 2484 919 2080 1613 1903 435 1338 1790 903 2903 1242 3338 612 1000 2354 9999 548 887 1661 1435 1532
1419 806 1967 1403 2855 903 3161 2080 3435 3500)
minneapolis__Minnesota(1758 2016 1806 1967 2645 1790 1338 1725 2387 2242 1532 2145 1419 661 1225 1951 1258
1532 1483 403 1112 2354 387 1871 2129 1903 951 1709 2354 709 2677 1338 3000 1145 1467 2855 548 9999 1338 2177
1967 1983 1274 612 2532 1935 2709 1435 2951 2016 3226 3193)
nashville__Tennessee(1064 1661 403 1983 2290 1129 2580 306 3193 1758 1161 871 2032 758 854 709 612 1064 1903
1112 871 2064 1790 2290 1613 1258 451 661 903 935 2919 564 3242 274 338 1467 887 1338 9999 854 1451 1080 1096
1225 1112 1274 2693 919 3580 1500 3226 3758)
new_Orleans__Louisiana(1774 2322 774 1935 2919 1838 2951 564 3467 2435 2032 1177 2225 1483 1709 1112 1483 838
2064 1580 1725 1774 2225 2371 2306 564 1290 338 903 1354 2790 677 3000 1129 661 1387 1661 2177 854 9999 2161
1677 1112 1661 1048 1983 2419 1838 3596 887 2967 3677)
new_York_City__New_York(387 258 1371 3226 822 322 3290 1580 4016 338 596 1242 2822 1306 758 1161 903 2516
2887 1806 1048 3467 2338 3290 193 2596 1177 1967 1532 1983 4145 2016 4500 1242 1774 2145 1435 1967 1451 2161
9999 596 2387 2016 1758 177 3951 612 4371 2935 4516 4726)
norfolk__Virginia(306 822 903 3048 1435 370 3355 1145 4064 935 967 709 2903 1403 822 629 903 2177 2855 1854 1177
3145 2371 3258 790 2177 1145 1532 1000 1871 4000 1645 4338 1048 1419 1564 1532 1983 1080 1677 596 9999 2209
2129 1242 435 3790 677 4500 2500 4322 4839)
oklahoma_City__Oklahoma(2145 2419 1338 871 3113 3113 1854 1161 2322 2725 1967 1838 1112 1338 1661 1709 1483
338 1016 887 1661 1096 1403 1322 2645 741 1193 903 1854 564 1790 548 2161 1209 774 2419 1419 1274 1096 1112
2387 2209 9999 741 1983 2242 1580 1790 2516 774 2145 2677)
omaha__Nebraska(1838 2113 1629 1435 3758 1854 1467 1419 2000 2371 1645 2048 806 774 1338 1903 1274 1064 871
209 1177 2016 693 1274 2209 1387 983 1403 2129 306 2096 935 2532 1129 1032 2645 806 612 1225 1661 2016 2129 741
9999 2274 1935 2177 1500 2371 1516 2629 2725)
orlando_Florida(1371 1983 693 2806 2613 1435 3596 887 4258 2096 1935 612 3016 1854 1693 709 1548 1774 3032 2193
1887 2709 2919 3338 1774 1580 1564 1129 225 2016 3790 1548 3919 1387 1258 370 1967 2532 1112 1048 1758 1242
1983 2274 9999 1596 3355 1580 4580 1887 3887 4629)
phidadelphia__Pennsylvania(209 419 1209 3145 951 161 3193 1403 3919 516 580 1064 2838 1274 693 983 774 2322 2806
1758 983 3338 2306 3209 338 2435 1064 1790 1371 1887 4000 1838 4355 1145 1629 1983 1403 1935 1274 1983 177 435
2242 1935 1596 9999 3822 500 4274 2806 4468 4677)
phoenix__Arizona(3709 4016 2951 741 4709 3726 1935 2709 1580 4306 3580 3419 1516 2806 3274 3274 3064 1613 1306
2306 3242 709 2725 935 4145 1871 2790 2354 3226 2000 467 2145 629 2822 2371 3790 2855 2709 2693 2419 3951 3790
1580 2177 3355 3822 9999 3371 1177 1613 564 1225)
pittsburgh__Pennsylvania(403 725 1096 2629 1387 403 2677 1274 3435 919 354 1048 2306 774 209 951 306 1951 2306
1258 483 2871 1822 2725 758 2209 580 1516 1338 1403 3564 1451 3919 629 1225 1903 903 1435 919 1838 612 677 1790
1500 1580 500 3371 9999 3822 2387 3935 4209)
reno__Nevada(4145 4451 3887 1645 5064 4226 1548 3677 693 4629 3951 4387 1564 3145 3645 4209 3564 2693 1661
2580 3548 1887 2548 1209 4516 3032 3322 3355 1338 2645 725 645 758 3451 3274 4829 3161 2951 3580 3596 4371 4500
2516 2371 4580 4274 1177 3822 9999 2790 967 370)
```

```
san_Antonio__Texas(2564 3177 1613 1177 3693 2629 2451 1387 2693 3258 2629 2064 1709 1951 2338 1983 2113 435
1532 1580 2338 903 2145 1758 3080 322 1919 1080 1742 1258 2080 2693 2242 1774 1177 2242 2080 2016 1500 887 2935
2500 774 1516 1887 2806 1613 2387 2790 9999 2096 2806)
san_Diego__California(4193 4597 3467 1306 5274 4322 2113 3209 1580 4645 4080 3903 1919 3371 3855 3838 3645 2177
1774 2855 3822 1177 3016 1371 4677 2403 3355 2838 3806 2564 548 3209 209 3371 2919 4322 3435 3226 3226 2967
4516 4322 2145 2629 3887 4468 564 3935 967 2096 9999 822)
san_Francisco__California(4580 4806 4000 1790 5451 4548 1919 3822 1048 5048 4306 4548 1919 3500 4000 4468 3919
2822 2032 2951 3871 1935 2935 1580 4968 3080 3693 3467 4435 3000 919 3629 629 3790 3419 4984 3500 3193 3758
3677 4726 4839 2677 2725 4629 4677 1225 4209 370 2806 822 9999)
)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%   Procesos del programa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  proc {Mostrar_Matriz Matriz Lista Ventana Op Vector Cantidad_a_mostrar}
    Cantidad = {Width  Matriz.1} %%Cantidad de Ciudades de la matriz de distancias
   {Ventana set(state:normal)}
   {For 1 Cantidad 1
          proc {$ A}
             {Ventana insert('end' "\n\n ")}
             {Ventana insert('end' Vector.A)}
             {Ventana insert('end' " ( ")}
             {Ventana insert('end' "\n ")}
             {For 1 Cantidad 1
              proc {$ B}
                if (B mod Cantidad_a_mostrar)==0 then
                       {Ventana insert('end' "\n")}
                end
if B \= A then
                {Ventana insert('end' Vector.B)}
                {Ventana insert('end' " : ")}
                {Ventana insert('end' Matriz.A.B)}
                {Ventana insert('end' " Kilometros ")}
                if B < Cantidad then
                       {Ventana insert('end' " - ")}
                end
                end
              end}
{Ventana insert('end' " ) ")}
          end}
    {Ventana insert('end' "\n\n")}
    if Op == 1 then
          {Crear_Variables Cantidad Lista}
          end
    {Ventana set(state:disabled)}
  end
proc{Crear_Variables Cantidad Lista}
Nombre_Ciudades = {MakeTuple ciudad Cantidad+1}
  Vector_Ciudades_modificada = {MakeTuple ciudad Cantidad}
     Matriz_Nueva = {MakeTuple filas Cantidad}
  {Lista_aux set(Lista)}
{For 1 Cantidad 1
 proc {$ A}
   Matriz_Nueva.A= {FD.tuple columnas Cantidad 0#9999}
 end}
end
proc{Nueva_Matriz_Modificada}
if Opcion == "Chile" then

   {For 1 Cantidad 1
    proc{$ P}
          if Vector_Chile.P == Opcion_Origen then
               Contador=P
             %{Browse {String.toAtom Vector_Chile.P}}

          end
    end}
   {Modificar_Matriz Matriz Matriz_Nueva Cantidad Contador 1}
    {Cambiar_orden_ciudades Vector_Chile Vector_Ciudades_modificada Cantidad Contador}
   {Mostrar_Matriz Matriz_Nueva Lista_Chile Texto2 2 Vector_Ciudades_modificada 4}
```

```
end
    if Opcion == "Brasil" then
          {For 1 Cantidad 1
           proc{$ P}

             if Vector_Brasil.P == Opcion_Origen then
              Contador=P
             end
           end}
          {Modificar_Matriz MatrizBrasil Matriz_Nueva Cantidad Contador 1}
          {Cambiar_orden_ciudades Vector_Brasil Vector_Ciudades_modificada Cantidad Contador}
          {Mostrar_Matriz Matriz_Nueva Lista_Brasil Texto2 2 Vector_Ciudades_modificada 4}
    end
    if Opcion == "Estados Unidos" then
{For 1 Cantidad 1
           proc{$ P}

             if Vector_USA.P == Opcion_Origen then

              Contador=P
             end
           end}

          {Modificar_Matriz MatrizUSA Matriz_Nueva Cantidad Contador 1}
          {Cambiar_orden_ciudades Vector_USA Vector_Ciudades_modificada Cantidad Contador}
          {Mostrar_Matriz Matriz_Nueva Lista_USA Texto2 2 Vector_Ciudades_modificada 3}
end
     {Boton2 set(state:disabled)}
    {Boton3 set(state:normal)}
end
proc{OpcionMatriz}
  if Opcion == "Chile" then
    {Mostrar_Matriz Matriz Lista_Chile Texto1 1 Vector_Chile 4}
  end
    if Opcion == "Brasil" then
          {Mostrar_Matriz MatrizBrasil Lista_Brasil Texto1 1 Vector_Brasil 4}
    end
    if Opcion == "Estados Unidos" then
          {Mostrar_Matriz MatrizUSA Lista_USA Texto1 1 Vector_USA 3}
    end
{Boton1 set(state:disabled)}
end
proc {Cambiar_orden_ciudades Orden_Original Orden_Nueva Cantidad Posicion}
  if Posicion > 1 then
    {For 1 Cantidad 1
         proc {$ Y}
                  if Y == 1 then
                    Orden_Nueva.1 = Orden_Original.Posicion
                  else
                    if Y < (Cantidad-Posicion+2) then
                       Orden_Nueva.Y = Orden_Original.(Posicion+(Y-1))
                    else
                      Orden_Nueva.Y = Orden_Original.(Y-(Cantidad-Posicion+1))
                    end
             end
end}
      else
    {For 1 Cantidad 1
     proc {$ I}
          Orden_Nueva.I = Orden_Original.I
     end}
      end
end

proc {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Col2}
                  Matriz_Nueva.Fila.Col =: Matriz_Origen.Fila2.Col2
end
```

```
proc {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion Fila Col Fila2 Cantidad_Aux Cont Aux}
 if Cont < Cantidad then
    if Posicion > Cantidad then
       if Aux < (Cantidad_Aux+1)  then
            {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Aux}
            {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux Cont
Aux+1}
            else
            {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux
Cantidad+1 Aux+1}
       end
   else
     {ModificarColumnaA Matriz_Origen Matriz_Nueva Fila Col Fila2 Posicion}
     {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion+1 Fila Col+1 Fila2 Cantidad_Aux-1 Cont Aux}
   end
 end
 end

proc {Modificar_Matriz Matriz_Origen Matriz_Nueva Cantidad Posicion P}
   if Posicion > 1 then
     {For 1 Cantidad 1
      proc {$ Y}
            if Y == 1 then
              {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 Posicion Cantidad 1 1}
            else
              if Y < (Cantidad-Posicion+2) then
                    {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 (Posicion+(Y-1)) Cantidad 1
1}
              else
                    {ModificarColumnaB Matriz_Origen Matriz_Nueva Cantidad Posicion  Y 1 (Y-(Cantidad-Posicion+1))
Cantidad 1 1}
              end
            end
          end}
else
   {For 1 Cantidad 1
    proc {$ B}
     {For 1 Cantidad 1
              proc {$ A}
                    Matriz_Nueva.B.A =: Matriz_Origen.B.A
              end}
   end}
 end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%   INTERFAZ
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
proc{SaveText}
   Name={QTk.dialogbox save(defaultextension:"txt"
                     initialdir:"C:/"
                     title:"Guardar Resultado del TSP..."
                     initialfile:""
                     filetypes:q(q("Archivo de Texto" q(".txt"))
                           q("All files" q("*"))) $)}
in
   try
     File={New Open.file init(name:Name flags:[write create truncate])}
     Contents={Texto3 get($)}
   in
     {File write(vs:Contents)}
     {File close}
   catch _ then skip end
end

 proc{LoadText}
%    Name={QTk.dialogbox load($)}
% in
```

```
try
            File={New Open.file init(name:"C:/TSP/"#Opcion#"/"#Opcion_Origen#".txt")}
      Contents={File read(list:$ size:all)}
   in
      {Texto3 set(state:normal)}
      {Texto3 set(Contents)}
      {Texto3 set(state:disabled)}
      {File close}
   catch _ then skip end
   {Boton3 set(state:disabled)}
    {Boton4 set(state:normal)}
 end
Inicio=lr(glue:we
        tdline(glue:wns)
label(text:"Seleccionar Matriz de Distancia (País): " glue:wnse
                        foreground: c(39 64 139)
                        background:c(155 205 155)
                    )
              tdline(glue:wns)
             Origen_Matriz
             button(text:"Aceptar " glue:we
                        %%Color botones en RGB
                        activeforeground:c(39 64 139)
                        activebackground:c(205 92 92)
                        foreground: c(39 64 139)
                        background:c(155 205 155)
%%%%
                    handle:Boton1
                state:disabled
                    action:OpcionMatriz)
             tdline(glue:wns)
            )
Orden =lr(glue:we
              tdline(glue:wns)
              label(text:"Seleccionar Ciudad de Origen: " glue:wnse
                        foreground: c(39 64 139)
                    background:c(155 205 155)

                    )
              tdline(glue:wns)
              Origen
              tdline(glue:wns)
              button(text:"Aceptar" glue:we
                        %%Color botones en RGB
                        activeforeground:c(39 64 139)
                        activebackground:c(205 92 92)
                        foreground: c(39 64 139)
                        background:c(155 205 155)
%%%%
                     handle:Boton2
                     state:disabled
                     action:Nueva_Matriz_Modificada)
             tdline(glue:wns)
)
 VentanaTSP=lr(glue:we
                tdline(glue:wns)
                button(text:"Calcular TSP" glue:we
                        %%Color botones en RGB
activeforeground:c(39 64 139)
                        activebackground:c(205 92 92)
                        foreground: c(39 64 139)
                        background:c(155 205 155)
                        %%%%
handle:Boton3
                        state:disabled
                        action:LoadText)
```

```
tdline(glue:wns)
                button(text:"Guardar" glue:we
                %%Color botones en RGB
                        activeforeground:c(39 64 139)
                        activebackground:c(205 92 92)
                        foreground: c(39 64 139)
                        background:c(155 205 155)
                        %%%%
                 handle:Boton4
                        state:disabled
                        action:SaveText)
                tdline(glue:wns)
                button(text:"Salir" glue:we
                        %%Color botones en RGB
                        activeforeground:c(39 64 139)
                        activebackground:c(205 92 92)
                        foreground: c(39 64 139)
                        background:c(155 205 155)
                        %%%%
                        action:toplevel#close)
                tdline(glue:wns)
               )
Origen=lr(label(handle:Lista_Valor
                    width:20
                    background:c(155 205 155)
                    foreground: red
                    glue:wns
                   )
            dropdownlistbox(.
                            buttonactivebackground:blue
                            buttonbackground:red
                            handle:Lista_aux
                    height:15
                    lrscrollbar:true
                    tdscrollbar:true
                    glue:wns
                   % {Lista_aux  set(state:disabled)}
                            action:proc{$}{Lista_Valor set({List.nth
                                                    {Lista_aux get($)}
                                                    {Lista_aux get(firstselection:$)}})}
                                    Opcion_Origen = {Lista_Valor get($)}
                                    {Boton2 set(state:normal)}
                                end)
          )
Origen_Matriz=lr(label(handle:Lista_Valor2
                    width:20
                    background:c(155 205 155)
                            foreground: red
                    glue:wns
                   )
            dropdownlistbox(init:["Chile" "Brasil" "Estados Unidos"]
                            buttonactivebackground:blue
                            buttonbackground:red
                            handle:Lista_aux2
                            height:3
                    glue:wns
                            action:proc{$} {Lista_Valor2 set({List.nth
                                                    {Lista_aux2 get($)}
                                                            {Lista_aux2 get(firstselection:$)}})}
                                    Opcion = {Lista_Valor2 get($)}
                        {Boton1 set(state:normal)}
                                end)
                    )
Interfaz=panel(glue:nswe
            td(
            title:"Seleccionar País"
```

Anexo: Código del Protótipo de Sistema com CSOP en Oz utilizando QTk

```
Inicio
                text(handle:Texto1 bg:white tdscrollbar:true lrscrollbar:true
                        height:20 width:120
              state:disabled
                        )
                 )
                td(
                title:"Seleccionar Ciudad de Origen"
                Orden
                text(handle:Texto2 bg:white tdscrollbar:true lrscrollbar:true
                        height:20 width:120
              state:disabled
                        )
               )
td(
               title:"Resolución del TSP"
               VentanaTSP
               text(handle:Texto3 bg:white tdscrollbar:true lrscrollbar:true
                        height:20 width:120
                        state:disabled
                   )
              )
                borderwidth:2
                        handle:AuxPanel
                  )
in
   Ventana={QTk.build td(Interfaz
                                  )}
    {Ventana set(title:"Prototipo Final de Tesis"
                         borderwidth:10
                        )}
   {Ventana show}
    {Ventana set(geometry:geometry(x:50 y:150))}
end
```