

**UNIVERSIDAD ANDRÉS BELLO**

**FACULTAD DE INGENIERÍA**

**INGENIERÍA CIVIL INFORMÁTICA**

**Trabajo 5 – Problem Solving**

**ANDRÉS EDUARDO VALENZUELA GONZÁLEZ**

**SANTIAGO - CHILE**

**NOVIEMBRE, 2017**

Contenido

[**1.** **Introducción a Lisp** 3](#_Toc496662006)

[**2.** **Introducción a Prolog** 3](#_Toc496662007)

[**3.** **Desarrollo del problema** 4](#_Toc496662008)

[**4.** **Analisis de Resultados** 4](#_Toc496662009)

[**5.** **Conclusiones** 13](#_Toc496662010)

# **Introducción**

El problema del vendedor viajero responde la pregunta: “dada una lista de ciudades conexas entre si y cada conexión posee un coste de viaje, ¿Cuál es la mejor ruta que conecte a todas las ciudades y represente el menor coste de viaje volviendo asi a la ciudad de origen?”. Dentro de la optimización combinatoria, este es un problema *NP-duro* (un problema del cual se provee una solución en tiempo polinómico).

Dada la presente problemática, se pueden proponer como solución diversas heurísticas (o bien diseñar alguna) tales como “Algoritmo del vecino mas próximo”, “Intercambio par a par”, “Heuristica de k-opt o heurística de Lin-Kernighan”, “*Bee Colony Optimization* (BCO)”, “*Ant Colony Optimization* (ACO)” o “Algoritmos genéticos”.

Para un algoritmo genético existen diversas implementaciones (seguir un modelo solución, ACO, BCO o diseñar uno propio), los cuales buscan recrear la sabiduría de la naturaleza en cuanto a creación de población, selección de población, evolución y/o mutación para luego repetir el proceso hasta encontrar la mejor solución. Dadas estas características, el algoritmo genético no siempre encontrara la solución óptima pero si una *óptima local*. Como fue mencionado antes, esto dependerá netamente de la implementación y logica que se le dé al algoritmo.

# **Desarrollo del problema**

Se debe implementar uno o más algoritmos para poder realizar problemas TSP de la biblioteca de problemas de la Universidad *Heidelberg*, que entrega un archivo .tsp con la información de los nodos y los caminos:

<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/>

Para este informe se solicita probar varias instancias de distintos tamaños y analizar los resultados en términos de calidad de la solucion versus tiempo de ejecución. Tambien entre números de iteraciones y tiempo de ejecución, probar distintos parámetros de la estrategia seleccionada de manera que optimice su desempeño, analizar las mejores soluciones encontradas a la luz de las mejores provistas por la biblioteca y por ultimo, proponer mejoras que se podrían hacer para mejorar el desempeño.

# **Analisis de Resultados**

Para el desarrollo de este informe el alumnado se vio en la libertad de diseñar su propio algoritmo genético (del caso contrario, comparar los algoritmos ACS y BCO).

En este caso, se propuso un algoritmo genético desarrollado en Python 2.X, orientado a objetos y multi-thread. La implementación cuenta con 3 archivos de los cuales en el archivo *Genetic.py* y *TSP.py* se pueden ubicar 5 clases distintas, las cuales proveen de bastante lógica aplicada para luego solo aplicar multi-threads en el archivo main.py para aprovechar la potencia de los nucleos y asi obtener mas de un resultado por ejecucion.

*TSP.py* clases:

# **Conclusiones**

conclusion