EL PROBLEMA DEL AGENTE VIAJERO CON VENTANAS DE TIEMPO

Informe

Jefferson Amado Peña Torres Julieth Alegria Vaca Cindy Valencia Tenorio
Universidad del Valle
Escuela de ingenieria en sistemas y computacion
November 22, 2014
Cali

***Abstract*—Este informe presenta una solucion al problema del agene viajero desde un modelo con restricciones y programacion linea,aplicados en el curso de Complejidad y Optimizacion 2014 y presentado a Irene Tisher y Nilso Mosso, profesores de la Escuela de ingenieria de sistemas y computacion de la Universidad del valle**

I. Descripcion del problema

Un agente viajero tiene que visitar n sitios de venta, cada uno una sola vez. Se conoce el tiempo de desplazamiento tdij entre cada dos sitios i y j . El agente viajero tiene que detenerse en cada sitio para visitar a sus clientes. El tiempo requerido en cada sitio i, denominado tiempo de servicio tsi y es conocido desde un comienzo. Un inconveniente en la vida real es que el agente viajero no puede visitar a sus clientes cuando ms le conviene a l, sino cuando su cliente lo puede atender. Por ello en cada sitio est denida una ventana de tiempo, [ai , bi] donde ai denota el momento ms temprano y bi el momento ms tarde en que el agente viajero puede ser recibido por el cliente en el sitio i. El agente viajero s puede llegar ms temprano que ai al sitio i pero tiene que asumir en este caso un tiempo de espera (no ser atendido antes de ese momento). El agente viajero no puede llegar ms tarde que bi al sitio i. El problema del agente viajero con ventanas de tiempo consiste en planear las visitas del agente dados los sitios y sus ventanas de tiempo, los tiempos de desplazamiento entre sitios y el sitio de origen minimizando el tiempo total requerido para atender a los clientes en los n sitios y regresar al sitio de origen, respetando las ventanas de tiempo.

II. Modelado del problema

Considere un conjunto de nodos **N=** $\{1, ...n, \}$ que seran visitados donde el nodo origen y el nodo destino seran el mismo nodo pues el *problema del agente viajero* **TSP** posee una estrecha relacion con el *Camino hamiltoniano* [13] que es definido como una sucesion de aristas adyacentes, que visita todos los vertices del grafo una sola vez.

El *problema del agente viajero con ventanas de tiempo* **TSPTW** es una variante del *problema del agente viajero* **TSP** que tiene asociado a cada nodo $i \in$ **V** una ventana de tiempo $w_i = [a_i, b_i]$, donde cada par de vertices estan unidos por una arista, es decir, contiene todas las posibles aristas **A** y cada arista tiene asociado un valor no-negativo que se el tiempo $td_{i,j}$.

El *problema del agente viajero con ventanas de tiempo* es el problema de encontrar una ruta de minimo costo visitando un conjunto de nodos solo una vez dentro de las ventana de tiempo descritas en cada nodo. El problema generalizado por el *problema del agente viajero* [12] es **NP-hard** y **Savelsbergh 1958** [1] mostro que encontrar una solucion factible del TSPTW es **NP-completo**

III. Formulacion matematica

En el modelo utiliza variables como: $X_{i,j}, (i,j) \in A$ que es una variable binaria y que indica indica el flujo entre los nodos $i \rightarrow j$ y s_i que es una variables de tiempo tal que $s_i \geq a_i \wedge s_i \leq b_i$. y la variable espera ε que adquiere valor cuando se llega a un nodo antes de que la ventana de tiempo $[a_i]$ sea abierta .

Para la formulacion del problema se requirio de dos nodos especiales o virtuales denominados "origen" **o** y "destino" **d**, sea **V=**$N \cup \{o, d\}$. Una permutacion permitida de nodos que se cumple todas las restricciones de las ventanas de tiempos, puede estar representada teniendo en cuenta los nodos artificiales **o** y **d**, que tienen tiempos de servicio $ts_o = 0$ y $ts_d = 0$ y tiempos de desplazamiento $td_{o,j} = 1$ y $td_{i,d} = 1$, donde el **o** posee aristas desde el hacia todos los nodos y todos lo nodos a **d**, estos nodos poseen ventanas de tiempo $W_o = [a_o, a_o]$ y $W_d = [a_d, a_d]$ donde $a_o = 0$ un valor menor que todos los a_i los otros nodos y $a_d = M$ un valor mayor que todos los b_j de los otros nodos. El objetivo a resolver es minimizar el tiempo de recorrido teninedo en cuenta el tiempo de servicio y el tiempo de desplazamiento denomindado como $Q_{i,j} = ts_i + td_{i,j}$.

La ruta de costo minimo inicia en el intervalo de tiempo que inicia en la ventana $[a_0, b_0]$ que tiene valores desde el archivo (0,0) como nodo origen, visitando exactamente una vez todos los nodos [12] que estan en **N** teniendo en cuenta las restriciones de las ventanas de tiempo en cada nodo se formula como:

$$X_{i,j} = \begin{cases} 1 & \text{Si el el nodo j es visitado despues de visitar i;} \\ 0 & \text{en otro caso.} \end{cases}$$

$$minimizar\ Z = \sum_{(i,j)\in A} Q_{i,j}X_{i,j} + \sum_{(i,j)\in A} \varepsilon_{i,j} \quad (1)$$

sujeto a:

$$\sum_{j=0,i\neq j} X_{i,j} = 1 \ \ \forall i \ \in \mathbf{V} - \{q\}\,, \quad (2)$$

$$\sum_{i=0,i\neq j} X_{i,j} = 1 \ \ \forall j \ \in \mathbf{V} - \{p\}\,, \quad (3)$$

$$u_i - u_j + NX_{i,j} \leq N - 1 \quad (4)$$

$$s_i + Q_{i,j} - e_i - s_j \leq (1 - X_{i,j})\, M_{i,j}, \quad (5)$$

$$a_i \leq s_i \leq b_i \,\forall i \ \in \ V, \quad (6)$$

$$s_j - Q_{i,j} + e_i - s_i \leq \varepsilon_{i,j} - M(X_{i,j} - 1), \quad (7)$$

$$X_{i,j} \in \{0,1\} \ \forall (i,j) \ \in \ A, \quad (8)$$

En (1) la variable $X_{i,j}$ es **1** si el arco (i,j) es usado en la ruta optima es **0** en otro caso, el valor $Q_{i,j} = ts_i + td_{i,j}$ es un numero que indica el valor entre (i,j) y $\varepsilon_{i,j}$es la espera generada en la arista (i,j).
Las restriccion (2) obliga a que desde un nodo solo se tome una arista hacia otro nodo, la (3) obliga a que desde otro nodo solo se tome una arista a este nodo, ambas aseguran que una ruta desde **o** hasta **d** sea un ciclo dirigido que pasa por todos lo nodos de **N** y que la variable $X_{i,j}$ con valor en 1 indique las transiciones $i \rightarrow j$ entre los nodos. La restriccion (4) impide que se generen ciclos, esta plantea un conjunto de ecuaciones donde cada valor u_i obliga a que se cubra todas las ciudades y no dos o ms caminos disjuntos.

la restriccion(5) aseguran que una solucion respete las ventanas de tiempo en cada nodo y que se asegure la factibilidad del tiempo planeado,cuando $b_{i,j} + Q_{i,j} \leq a_{i,j}$, es decir, el mejor de los caso las restricciones siempre seran satisfechas para cualquier valor de s_i y de s_j, para el otro caso puede que no halla solucion,la restriccion (6) asegura que el momento de inicial es servicio en el nodo i este dentro del marco de la ventana de cada nodo **N** haciendo que sea factible o no la solucion,la restricion (7) modela la espera, si se llega a un nodo *i* antes de la apertura de la ventana se debe esperar y el tiempo gastado en ese nodo sera la suma

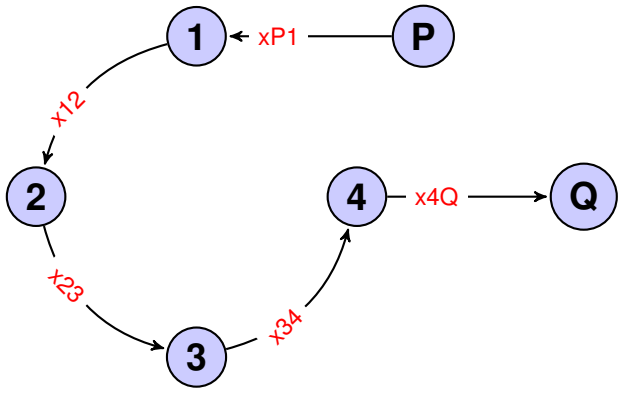
la restriccion (8) indica que la variable $X_{i,j}$ solo puede tomar valores 0 y 1 (**binaria**)

A. *Estrategia de los nodos artificiales*

Los nodos artificiales permiten una solucion factible desde un nodo **i** a el mismo **i**, pues es el unico ciclo permitido (*Hamilton path* [13]) las variables $X_{o,j}$ y $X_{i,d}$ estaran en la solucion permitiendo que desde el nodo origen se pueda ir a cualquier otro nodo, pero el no es alcanzable para ninguno, y el destino alcanzable para todos pero que de el no se puede continuar con la ruta pues es el destino

La importancia de estos nodos es que cualquier nodo puede tener como ventana de tiempo $W_i = [0,0]\ \forall i \in N$ el modelo podra indicar que la ruta debe iniciar por este nodo i y debe terminar en el mismo.

Si la ruta optima inicia por el nodo con id 1, dado que la ventana en este punto sea $W_1 = [0,0]$ y supuniendo la ruta que se describe en el grafico la ruta para el lpsolve sera $P \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow Q$, donde xP1 y x4Q tiene valor 1 y son asu vez el nodo por donde se inicio y donde se termino la ruta es decir 1



El haber agregado estos nodos acarreo con algunas sumas en el valor objetivo, pues como la ruta verdaderamente no va al nodo de origen, las aristas **xP*** y **x*Q** tiene un $Q_{i,j} = 1$, es decir se suma un al valor objetivo y no el valor real de la arista.

IV. IMPLEMENTACION

El **Univalle TSPTW Solver** es un aplicativo que implementa el modelo propuesto en este informe, creando de manera dimanica las restricciones segun la instancia de entrada y utilizando las librerias de **lpsolve** [15] con las que genera una solucion al problema TSPTW.
El codigo fuente esta escrito en Java (lenguaje de programacion) [14] y se encuentra adjunto a este informe,**Univalle TSPTW Solver** es imlementado para **linux** y usa **Make** para la gestion de dependencias.

Univalle TSPTW Solver se ha probado y comparado con las instancias de Dumas Benchmarks [4] y Silva-Urrutia Benchmarks [2] creando una lectura y transformacion de instancias a las propuestas en el proyecto, obteniendo una solucion optima para 15 nodos La carpeta principal de **Univalle TSPTW Solver** tiene las subcarpetas: **entrada:** Aqui reside algunos archivos de entrada para el aplicativo **informe:** Aqui reside el informe y las graficas obtenidas de las implementacion **src:** Aqui reside el codigo fuente en tres subcarpetas, algorithms que contiene los algortimos que generan las restricciones y el uso del API de lpsolve, data que contiene la lectura del archivo y el acceso a los datos y view que contiene la interfaz grafica de usuario(GUI). **salidas:** Aqui reside la soluciones a las instancias en formato de texto plano **lib:** Aqui reside las librerias de lpsolve para arquitecturas de 32-bits y 64-bits y el (.jar) que contiene el API de lpsolve **xlpsolve:** algunos archivos en formato (.lp) que sirven para ver el modelo en ejecucion **doc(opcional):** Aqui reside si se genero documentacion el API de **Univalle TSPTW Solver class(opcional):** Aqui reside si se compilo los (.class) o bycode generado por java **bin(opcional):** Aqui reside

si se genero un (.jar) con el ejecutable de **Univalle TSPTW Solver**

1) compilacion: **Univalle TSPTW Solver** utiliza **Make** en una emulador de terminal ubicandose en la carpeta del proyecto se teclea **make** y este comando generara lo necesario para su ejecucion

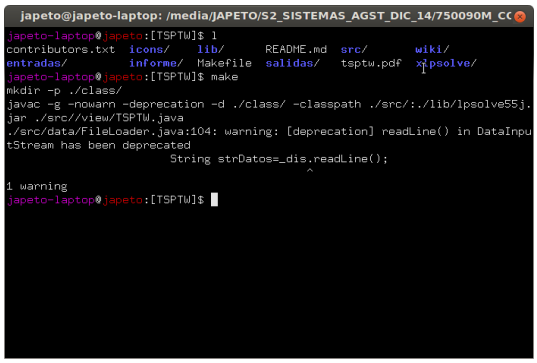


Fig. 1: compilacion en xterm usando el comando make

2) ejecutar Univalle TSPTW Solver: **Univalle TSPTW Solver** utiliza **Make** en una emulador de terminal ubicado en la carpeta del proyecto se teclea **make run** o **make debug** ejecutara el aplicativo en modo simple en el que solo entrega resultados o en modo depuracion que muestre en el emulador de terminal las interaciones que va haciendo paso a paso

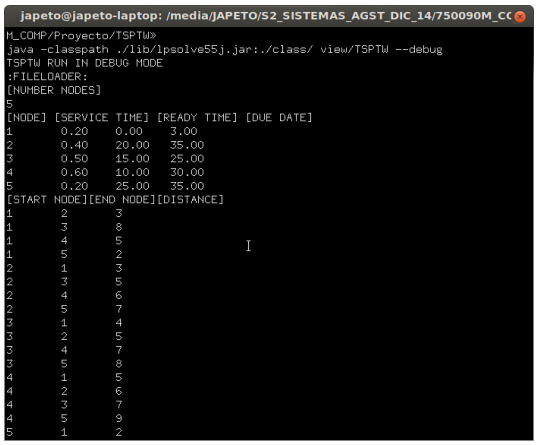


Fig. 2: modo depuracion

3) interfaz grafica de usuario GUI: Para cargar una nueva instancia se debe pulsar el boton de abrir (open) y se mostrara un grafo totalmente conectado,es decir de un nodo a todos, cada nodo presentara un color azul y con un texto **c:ID time-service** $[a_i, b_i]$ este se podra mover con el raton y su ubicaion es aleatoria, las aristas entre los nodos puede ser de varios colores y encima de estas aparece el tiempo de viaje entre un par de nodos $td_{i,j}$, el numero de nodos y arista en el archivo se puede ver en la parte inferior derecha en espacio log.

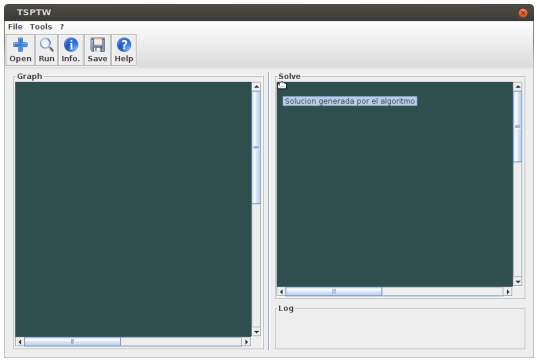


Fig. 3: interfaz grafica de usuario GUI

4) Puesta en marcha: **Univalle TSPTW Solver** despues de cargar una instancia desde un archivo se puede ejecutar el algortimo desde el boton run y este generar en log los nodos que hacen parte de la solucion y el valor minimo obtenido en rojo, ademas mostrara en la parte superior derecha el grafico solucion

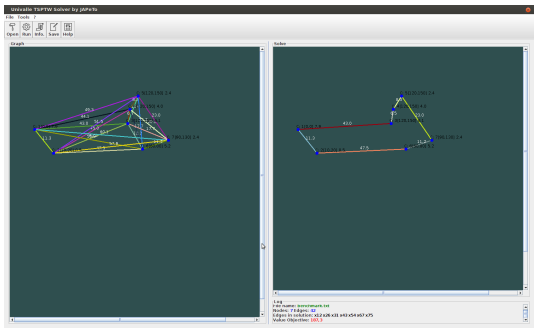


Fig. 4: modo depuracion

5) documentacion: **Univalle TSPTW Solver** esta escrito en **Java** y utiliza **Make** en una emulador de terminal ubicado en la carpeta del proyecto se teclea **make doc** que genera la documentacion API por medio del javadoc y este queda en la subcarpeta ”doc”.

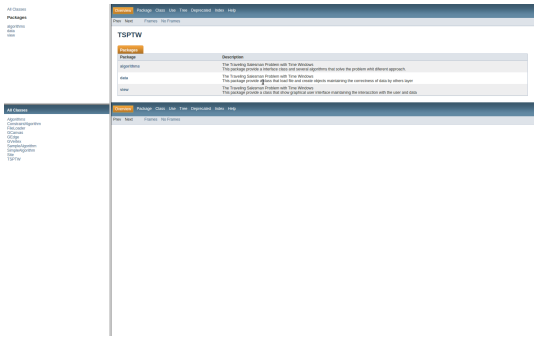


Fig. 5: Documentacion (API) de Univalle TSPTW Solver

V. Experimentos y pruebas

A. Solucion

La respuesta a una entrada es mostrada por **Univalle TSPTW Solver** en la parte inferior derecha, para una instancia primero el nombre del archivo de entrada,caracteristicas del

archivo, como el numero de nodos y aristas, luego los IDs de los nodos con las rutas entre los nodos que hacen minima la solucion.
para una instancia cualquiera la forma de la solucion sera:

```
(n20wX.001.txt)(162.12)
1 -> 2, 3->6, 2->4,6-> 10,10->9,8->7,
P->2, 5->Q
```

y que podra ser guardada como un archivo de texto plano, se incluye los nodo artificial que indica por donde inicio la ruta,$P \rightarrow 2$ inicio por el nodo 2 , y el nodo artificial que inidica cual es el nodo antecesor al nodo destino, $5 \rightarrow Q$ antes de ir a dos pasa por 5

1) Algunas instancias: Para pruebas utilizamos algunas entradas de 4,5,6,7,8,9 nodos con ventanas factibles y de multiples tamaõs, ademas se probo con los archivos entregados por la profesora mencionados en la tabla como tsptw1.txt y tsptw2.txt. Para el cálculo de los tiempos.
La siguiente tabla muestra los tiempos que se tarda en generar las restricciones **Univalle TSPTW Solver** y el tiempo que tarda el **LpSolver** en resolver la matriz y obtener el valor objetivo, todos estos archivos se encuentran en la carpeta entrada.

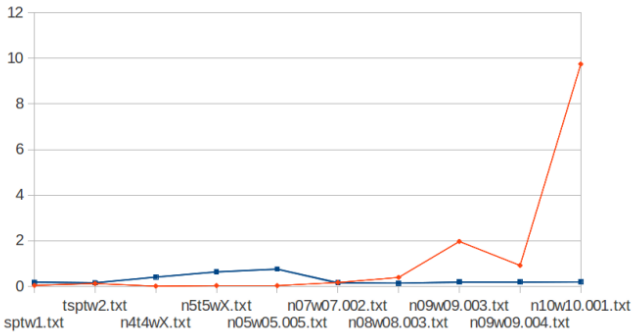


Fig. 6: Gráfico de los datos

Archivo	tiempo Restricciones	tiempo solver
tsptw1.txt	0.173ms	0.035ms
tsptw2.txt	0.149ms	0.115ms
n4t4wX.txt	0.040ms	0.005ms
n5t5wX.txt	0.063ms	0.026ms
n05w05.005.txt	0.075ms	0.023ms
n07w07.002.txt	0.153ms	0.162ms
n08w08.003.txt	0.129ms	0.385ms
n09w09.003.txt	0.189ms	1.964ms
n09w09.004.txt	0.187ms	0.911ms
n10w10.001.txt	0.195ms	9.747ms

Como puntos de referencia, se utilizaron dos formatos que en la mayoria de lecturas realizadas se toman como referente, **Univalle TSPTW Solver** posee un convertidor desde estos formatos con el fin de utilizarlos como pruebas para el modelo, dando credito a sus respectivos autores, que toman gran cuidado en mantener un formato para las instancias de manera uniforme, esto autores y sus formatos son:

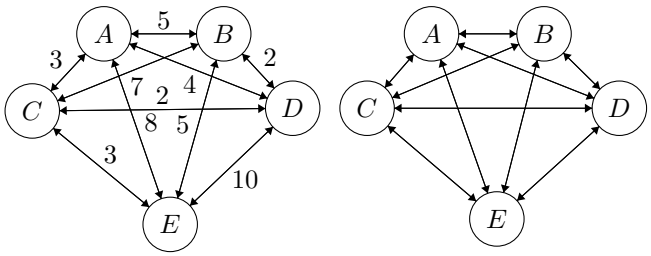
2) Dumas Benchmarks [4]: Un archivo o instancia presentada en el formato Dumas Benchmarks [4] contiene, primero un numero que indica el numero de nodos (n) incluido en nodo destino, luego posee posee una matrix de adyacencia (normalmente son los tiempos de viaje $td_{i,j}$ entre los nodos $i \neq j$), finalmente se encuentran las ventanas de tiempo para cada nodo estos fieron tomados de la pagina de sus Benchmarks [19]

3) Silva-Urrutia Benchmarks [2]: Un archivo o instancia presentada en el formato Silva-Urrutia Benchmarks [2] contine,una linea comentada con el nombre del archivo, luego unas etiquetas CUST NO., XCOORD, YCOORD,DEMAND,[READY TIME],[DUE TIME] y [SERVICE TIME] que acertadamente concuerdan con las necesidades del TSPTW,aunque el autor no usa la columna DEMAND y en sus archivos no piensa tener mas de 999 nodos pues este numero define el limite de los nodos estos fieron tomados de la pagina de sus Benchmarks [20]

VI. Resultados computacionales

El modelo descrito codificado en java se ejecuto en un Intel Pentium Dual-Core Mobile T4200 a 2,0 GHz de arquitectura de 32-bits y con la versión 7 de java, En la practica el problema de agente viajero con ventanas de tiempo es dificil, es decir puede verificarse en tiempo polinomial pero no exite un algoritmo que halle una solución poco tiempo dependiedo de la instancia.

A. Complejidad computacional



EL TSP es **NP**, para el grafo de la derecha el TSP sera encontrar una ruta que pase por todos los vértices una sola vez y que la suma del recorrido sea el menor, en el gráfico de la derecha un ciclo hamiltoniano es un ciclo que pasa por todos los vértices una sola vez. Para probar que es el TSP es **NP** primero debemos probar que dada una ruta y un valor de recorrido podemos facilmente ver si es ruta y si de verdad es un valor mínimo en una ruta, por ejemplo la ruta P=A,B,C,D,E,A con uan longitud de 32 y en tiempo polinomial se puede verificar si es respuesta a una instancia de TSP, Ahora es NP-duro si $CH \leq_p TSP$. Una instancia de ciclo Hamiltoniano (CH) G = (V,E) y se crea una instancia de TSP G'=(V,E') donde

$$E' = i,j \begin{cases} 1 & Si \quad (i,j) \in E \\ 0 & Si \quad (i,j) \notin E \end{cases}$$

. Ahora probamos que G tiene un camino Hamiltoniano si y solo si G' tiene un ruta de longitud 0 entonces TSP es

NP-completo. Las ventanas de tiempo hacen el problema considerablemente difícil, es decir el TSPTW es **NP**, para probar la afirmación debemos tener en cuenta que 3-SAT, 4-SAT y M-SAT [17] es NP-completo, ahora se reducira M-SAT a TSPTW, sea una instancia de M-SAT con las variables $v_i(x); 1 \leq i \leq n$ que tendran valores "falso" o "verdadero", segun $x = 1, 2, \cdots, X$ y las clausulas $D_i, \cdots, D_k$ donde cada clausula tendrán la forma $D_k = [\, v_a(x)\, OR\, v_b(x)\, OR\, v_c(x)]$, el tiempo de la apertura de la vantana a_i en el nodo *i* sera $v_i(x)$ y el tiempo de cierre de la ventana b_i en el nodo *i* sera $\neg v_i(x)$ y $v_i(x)$ es falso o verdadero dependiendo si el agente llego o no llego al nodo a cumplir con el servicio al abrirse o al cerrarse la ventana, si llega dentro del marco de la ventana, es decir, llega en un tiempo mayor a a_i y menor que b_i el $v_i(x) = *$

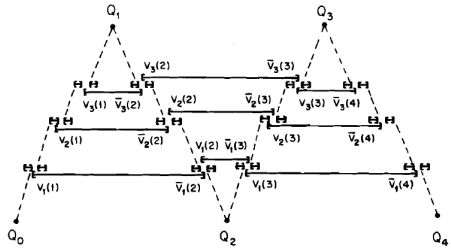


Fig. 7: Demostración de complejida en [18]

El anterior gráfico se puede escribir en CNF como $(v_1(1) \vee v_2(1) \vee v_3(1)) \wedge (\neg v_1(2) \vee \neg v_2(2) \vee \neg v_3(2)) \wedge (v_1(2) \vee v_2(2) \vee v_3(2)) \wedge (\neg v_1(3) \vee \neg v_2(3) \vee \neg v_3(3)) \wedge (v_1(3) \vee v_2(3) \vee v_3(3)) \wedge (v_1(4) \vee v_2(4) \vee v_3(4))$ otras formas de demostracion de complejidad estan descritas en articulo *Special Cases of Traveling Salesman and Repairman Problems with Time Windows* [18]

VII. Conclusiones

En este proyecto se presento una solución para el problema del Agente Viajero con Ventanas de Tiempo, una generalización conocida del TSP clásico en la que cada nodo debe ser visitado dentro de una ventana de tiempo determinada con el fin de obtener una ruta que genere un mínimo costo.

En este modelo se ha demostrado que la programación con restricciones puede ser usada para resolver problemas de combinatoria como lo es el TSPTW. Quizas la importancia del proyecto radica en la capacidad de abordar un problema y proveer una solución

En particular, se establecio un modelo general sobre la base de la partición de las ventanas de tiempo y el desplazamiento generado en cada nodo. Mostramos como implementar esta idea para obtener una formulación sólida e incorporarlas en un marco de ramificación clásico.

Algunas instancias generan rutas casi exclusivas, las ventanas de tiempo son estrictas si sus intervalos de tiempo son pequeños y en algunos casos de prueba se superponen entre los sitios, de acuerdo a lo anterior,la experimentación arrojó que las soluciones dependen de lo amplio o estrecho que sea el marco de la ventana y hay casos donde son estas que hacen que se pueda hallar una solucion factible o no.

Para el problema del agente viajero con ventanas de tiempo, se puede encontrar amplia literatura acerca de este, sobre todo algoritmos relativamente nuevos, dado que es un problema muy complejo de resolver, no se utilizan técnicas complejas en la actualidad, dado que se necesitan buenas soluciones en tiempos "acotados" (resolución en tiempo polinomial), a futuro se ve cambios en mutación entre técnicas, para ver las calidades de las soluciones implementadas pero ninguna resuelve el problema en tiempo polinomial. [18]
La libreria **LPSolve** nos permite solucionar un problema de modelado como este pues cuenta con lo necesario para adicionar las restricones y la variables ademas de solucionar la matriz resultante y obtener el valor objetivo

References

[1] M.W.P. Savelsbergh. Local search in routing problems with time windows, *Annals of Operations Research 4*, 285305, 1985. http://oai.cwi.nl/oai/asset/2449/2449A.pdf
[2] R. Ferreira da Silva and S. Urrutia. A General VNS heuristic for the traveling salesman problem with time windows, Discrete Optimization, V.7, Issue 4, pp. 203-211, 2010.
[3] Focacci F., Lodi A., Milano M."'Embedding relaxations in global constraints for solving TSP and TSPTW"' *Annals of Mathematics and Artificial Intelligence*, vol. 34, 2002,pp. 291311.
[4] Y. Dumas, J. Desrosiers, E. Gelinas and M.M. Solomon, An optimal algorithm for the travelling salesman problem with the time windows, Oper. Res. 43 (1995) 367371.
[5] Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut http://www.zib.de/groetschel/pubnew/paper/ascheuerfischettigroetschel2001.pdf
[6] Embedding Relaxations in Global Constraints for Solving TSP and TSPTW http://www.jcyl.esjcylceedgeaecongresos_ecoregCERCL32702.PDF
[7] An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows http://www.crt.umontreal.ca/~quosseca/pdf/56-TranspSci.pdf.
[8] A Time Bucket Formulation for the TSP with Time Windows http://www.optimization-online.org/DB_FILE/2009/11/2452.pdf.
[9] A Hybrid Exact Algorithm for the TSPTW http://www.or.deis.unibo.it/research_pages/ORcodes/FocacciLodiMilano.pdf.
[10] Solving VRPTWs with Constraint Programming Based Column Generation http://w1.cirrelt.ca/~louism/RGP_VRPTW_HCG.pdf.
[11] the TSP with time windows https://or-tools.googlecode.com/svn/trunk/documentation/user_manual/manual/TSP.html
[12] Travelling salesman problem http://en.wikipedia.org/wiki/Travelling_salesman_problem
[13] Hamiltonian path http://en.wikipedia.org/w/index.php?title=Hamiltonian_path&oldid=621067399
[14] Java (lenguaje de programacin) http://en.wikipedia.org/wiki/Java_(programming_language)
[15] Mixed Integer Linear Programming (MILP) solver http://lpsolve.sourceforge.net/5.5/index.html
[16] Traveling Salesman Problem with Time Windows http://www.eio.uva.es/~jsaez/maio/tsptw.pdf.
[17] Boolean Satisfiability Problem http://en.wikipedia.org/wiki/Boolean_satisfiability_problem.
[18] Traveling Salesman Problem with Time Windows Complexity http://www.mit.edu/~jnt/Papers/J037-92-TSPTW.pdf.
[19] Instance by Traveling Salesman Problem with Time Windows http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm.
[20] This is a not exaustive list of benchmark instances for TSPTW. http://homepages.dcc.ufmg.br/~rfsilva/tsptw/#instances.