Una Metaheurística para el Problema de Ordenamiento de Flota y Asignación de Tripulación en Sistemas de Transporte Público

---

Federico Bello ● Miguel Machado ● Juan Vignola

Proyecto de Grado

Ingeniería en Computación

Departamento de Investigación Operativa,
Instituto de Computación,
Facultad de Ingeniería,
Universidad de la República,
Montevideo - Uruguay

2010

Tutores
Msc. Ing. María Urquhart
Msc. Ing. Antonio Mauttone

Resumen

En torno al Transporte Público Urbano y Suburbano existen múltiples problemas a resolver, los cuales en general, revisten un alto grado de complejidad. Los actores principales allí son las entidades reguladoras, las compañías de transporte, sus flotas vehiculares, tripulación y pasajeros. Las entidades reguladoras se responsabilizan de la planificación estratégica del Transporte Público mientras que las compañías realizan la planificación operacional de su subsistema.

En este proyecto se estudiará el problema operacional particular de las compañías de ómnibus relacionado con el ordenamiento y asignación de su flota y tripulación, de modo de cubrir los recorridos asignados a la compañía de la mejor manera posible, desde el punto de vista de costos operacionales. La correcta resolución de este problema tiene enorme implicancia en la economía tanto de las empresas que brindan el servicio como en la de los pasajeros. Tratándose de transporte público las empresas competirán buscando proveer a la población con un servicio de buena calidad que les permita transportarse por tarifas acordes, lo que requiere una buena administración y organización de recursos de alto costo como son los vehículos, tripulación, mantenimiento de depósitos y flota, para que tanto las empresas como pasajeros se vean beneficiados.

El núcleo del problema a resolver consiste en: dados un conjunto de viajes a realizar, una flota heterogénea de vehículos disponibles distribuidos en múltiples depósitos y un conjunto de reglas laborales, determinar los libros de servicio. Es decir, la asignación que debe realizar cada vehículo y la tripulación para cubrir los viajes estipulados, respetando las restricciones y reglas laborales existentes, procurando minimizar los costos definidos. Con este fin, se plantea el estudio de una metaheurística para el diseño de un algoritmo y desarrollo de un sistema para la planificación óptima de sistemas de transporte público colectivo, así como también el diseño y desarrollo de un módulo de reglas laborales que permita la definición de dichas restricciones y validación de las soluciones generadas. Se presenta además, un análisis de los resultados obtenidos, utilizando casos de estudio significativos.

Palabras Clave: Transporte Público, Ordenación de Vehículos, Asignación de Tripulación a Flota, Reglas Laborales, Algoritmo Genético, Algoritmo Evolutivo, Metaheurística.

Índice

1. Introducción

Los problemas relativos al transporte público han sido abordados por investigadores[DESAUL] de todo el mundo, utilizando diversos enfoques y aplicándose a un conjunto amplio y heterogéneo de realidades y escenarios. Es así, que se ha establecido la existencia de múltiples etapas que forman parte del proceso de generación del ordenamiento y asignación eficiente de recursos. Cada una de ellas podrá considerar distintos factores, comunes o particulares, que definirán tanto la complejidad de resolución de cada etapa como del problema global influyendo considerablemente en las posibles soluciones.

La primera de las etapas es la denominada *Planeamiento Estratégico*. En dicha etapa el problema a resolver consiste en diseñar rutas y redes de transporte público. Se toman en cuenta, como datos fundamentales, la demanda de los usuarios de viajes de un punto a otro y el diseño de la ciudad (rutas, calles, avenidas, etc.). Normalmente, las rutas no son modificadas con gran frecuencia, por lo cual, la etapa de planeamiento estratégico se realiza una única vez, y por mucho tiempo no se vuelve a realizar. La mayoría de las veces el planeamiento estratégico es realizado a través de alguna entidad reguladora o planificadora, éste es el caso en el cuál, por ejemplo, el estado licita la concesión de un conjunto de líneas y la empresa las acepta, o no, dadas sus expectativas de recupero de las inversiones.

La siguiente de las etapas correspondientes al planteo anterior, es la de *Planeamiento Táctico*. En esta etapa, ya se consideran las rutas como determinadas, y lo que se debe obtener es la frecuencia con la cual se servirá cada una de ellas y una tabla de horarios (*timetable*) que es el producto final de ésta etapa. El dato relevante para ésta etapa es la demanda de los usuarios. El planeamiento táctico se suele hacer una vez para cada estación del año, y se suelen generar tablas de horarios diferentes para los días de semana y feriados. Por ejemplo, una tabla de horarios podría aplicarse para los días feriados en verano, y otra para los días de semana en otoño. Ocasionalmente, también se podrían generar tablas de horarios diferentes para los horarios del día y la noche.

Finalmente, la última etapa de planificación en el transporte público es la etapa de *Planeamiento Operacional* o *Planeamiento Operativo*. En ésta etapa, dado un conjunto de viajes con origen, destino y horario definidos, un conjunto de recursos disponibles, como ser flota y tripulación, así como su distribución en los diferentes depósitos disponibles, la entidad que pretenda brindar los servicios buscará asignar dichos recursos a los viajes programados en las etapas anteriores, de manera de minimizar los costos.

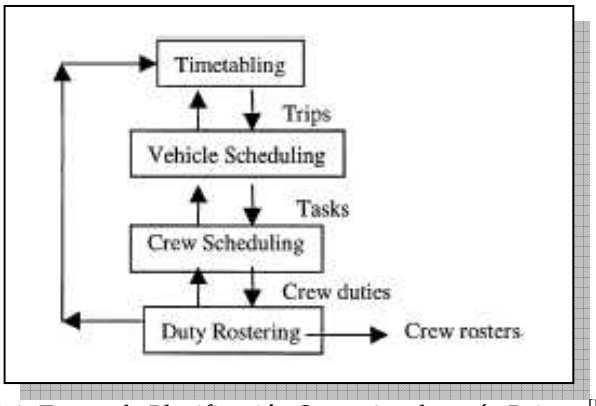

Fig. 1.1. Etapas de Planificación Operacional, según Paixao [PAIX]

La asignación de los vehículos de la flota (*Vehicle Scheduling*), en la última etapa mencionada, es clave a la hora de determinar uno de los principales costos en los que se puede incursionar haciendo una asignación ineficiente como lo es la cantidad de kilómetros de viajes "expresos". En la sección 3.7 donde se especifica el problema se definirán formalmente cada uno de los conceptos involucrados incluyendo viaje expreso, pero por ahora bastará conocer que es un viaje en el cual no se transportan pasajeros por lo que no produce ningún tipo de ingreso que permita cubrir sus costos.

La asignación de la tripulación (*Crew Scheduling*, habitualmente conductores de los vehículos) también es un problema que se aborda en la etapa de planificación operacional, dicha asignación, normalmente busca minimizar tanto la cantidad total de horas trabajadas, así como la cantidad de jornales de personas requeridas para cubrir dicho trabajo. Asimismo, la asignación de personal debe cumplir, habitualmente, con diversas restricciones de seguridad, y acuerdos colectivos o reglas laborales, tales como una máxima cantidad de horas para el jornal de trabajo, o el cumplimiento de una determinada cantidad de tiempo de descanso en la jornada. Finalmente, se realiza la etapa de *Rostering* donde se designan las personas para cubrir los turnos generados en la etapa anterior.

Los problemas de asignación de flota y tripulación, han sido abordados de manera separada, resolviéndose así dos problemas sucesivos, uno en el cual se asigna la flota de vehículos a los viajes (*Vehicle Scheduling Problem*, VSP[DESAUL]), y otro en el cual se asigna la tripulación a la flota ya asignada (*Duty Scheduling Problem*[DESAUL], *Bus Driver Scheduling Problem*[PAIX]). Utilizando éste enfoque, se han obtenido resultados[PAIX], sin embargo, no es siempre cierto que la mejor asignación vehicular, conlleve los mejores costos en cuanto a la tripulación a ser asignada luego. Fue en los años ochenta cuando se comenzó a contemplar la resolución del problema integrado fomentado por críticas al enfoque secuencial como la planteada por Ball et al (1983) [BALL]. De todos modos, algunas de las propuestas integradas realmente eficientes no aparecieron hasta 1995, como la planteada por Freling et al[IAVCS] y Gaffi, Nonato (1999)[IAECVSP]. Entre las formulaciones más recientes (2008) y ajustada al enfoque de la resolución propuesta en este trabajo, se encuentra la realizada por Jin-Kao Hao, Benoit Laurent (2008)[HAO08] donde su resolución se basa en técnicas como GRASP (*Greedy Randomized Adaptive Search Procedure)* e ILS (*Improvement by Local Search).*

En el presente trabajo, se estudia e implementa un Algoritmo Genético[GOLD] como estrategia de resolución. El mismo aborda el problema de planificación operacional, de forma integrada, es decir, se asignan simultáneamente, vehículos y tripulación, obteniéndose así un problema de mayor complejidad, pero con posibilidades de ofrecer resultados de mayor calidad. Para la asignación de flota, se consideran además restricciones que contemplan el uso de una flota heterogénea y viajes de distintas categorías que deben ser respetadas. Adicionalmente, para la tripulación, se dispondrá de un subsistema de definición de reglas laborales, las cuales, serán también restrictivas a la hora de realizar las asignaciones [DESAUL].

1.1. Objetivos

La creación de una solución que permita la obtención de libros de servicio buscando minimizar costos operativos, considerando una realidad modelada y definida a través de parámetros, restricciones y reglas, representa el principal objetivo del proyecto.

Durante el desarrollo y luego de finalizado el mismo, se han contemplado y alcanzado otros objetivos, los cuales serán abordados con mayor profundidad en los anexos A (Gestión y Desarrollo del Proyecto), B (Módulo de Algoritmo Genético) y C (Módulo de Reglas Laborales). Dichos objetivos pueden ser agrupados en dos categorías u objetivos generales. Uno de estos consta del diseño y generación de la estrategia para procurar alcanzar una resolución de buena calidad a un problema *NP-hard*[NPHARD] aplicando metaheurísticas[METAH], en particular, algoritmos genéticos[GOLD]. El segundo, agrupa los fines e intereses correspondientes al sistema a desarrollar. Se buscará crear, extender y aplicar al problema, una solución (*software*) teniendo entre sus principales requerimientos la flexibilidad y extensibilidad del domino, técnicas y procesos que componen el producto final, así como también su integración a una interfaz gráfica ya desarrollada[IUIO].

2. Marco Conceptual

El planeamiento operacional en una empresa de transporte público consiste principalmente de la asignación de recursos para cubrir los viajes previamente establecidos.

Los viajes, se dividen en dos categorías, viajes activos (*active trips*), y viajes expresos (*deadhead trips*). Los viajes activos, son los viajes en los cuales se transportan pasajeros. Estos viajes, no representan un costo, sino una ganancia para la empresa, debido a que, se supone, que los pasajeros abonan pasajes para viajar. Por otro lado, los mismos deben cumplirse, por lo cuál es imposible reducir costos al evitar realizar estos viajes.

Los viajes expresos (*deadhead trips*) son viajes que deben realizarse para cumplir con la cobertura de todos los viajes activos. Los viajes expresos, normalmente se clasifican, en tres categorías diferentes: *Pull out trips*, *Pull in trips*, y *deadhead trips*. Los viajes *Pull out*, son los que se realizan cuándo un vehículo tiene como localidad de origen de su primer viaje, un lugar distinto del sitio en dónde se encuentra el depósito. El viaje *Pull in* es lo inverso, es decir, el viaje que se realiza para guardar el vehículo en el depósito, luego de que el mismo haya cumplido con todos los viajes planificados en esa jornada. El viaje *deadhead* es un viaje expreso que se realiza entre la localidad de destino de un viaje activo y la localidad de origen del siguiente para un determinado vehículo. La distinción entre los tres tipos de viajes expresos, si bien aparece formalmente definida en la bibliografía [DESAUL] no es tan importante a la hora de determinar los costos en los cuales se incurre realizándolos, en los tres casos, el viaje expreso es clave a la hora de reducir costos, por lo que se debería evitar su realización tanto como sea posible.

La asignación de la tripulación a estos viajes, además de la asignación de flota es un trabajo muy complejo, debido a que siempre la asignación debe cumplir con las reglas laborales acordadas y, a la vez, optimizar el uso de recursos por parte de la empresa.

2.1. Planificación Operacional en Sistemas de Transporte Público

Para las empresas de transporte, la flota vehicular es uno de sus principales recursos, por tanto, no debe ser utilizado de manera ineficiente. Por un lado se busca minimizar el tamaño de la flota requerida (utilizar la menor cantidad posible de vehículos para cubrir todos los viajes) y también se busca evitar o minimizar que los vehículos realicen viajes improductivos (viajes expresos - *deadhead trips*). El problema VSP (*Vehicle Scheduling Problem)*[DESAUL] estudia y plantea cómo realizar la asignación vehicular.

La tripulación vista como un recurso, ésta vez humano, también es fundamental para la operativa de las empresas. Esta genera importantes costos para las mismas, por lo tanto debe ser utilizado eficientemente. Los objetivos fundamentales a ser resueltos en un problema de asignación de tripulación son, minimizar la cantidad de jornales

(cantidad de funcionarios que deben concurrir a trabajar en el día) y minimizar la cantidad de horas a pagar a cada uno de ellos para cubrir los viajes. El problema DSP (*Duty Scheduling Problem)*[DESAUL] estudia esta asignación.

Además, en el caso de la tripulación, otros objetivos deben ser contemplados, uno de ellos y que además va a ser parte importante del presente trabajo, es el de cumplimiento de las normas laborales vigentes.

Las reglas laborales, son acuerdos realizados entre cada funcionario (o un conjunto representativo de los mismos) y la empresa, además, habitualmente deben estar de acuerdo a leyes y/o normativas vigentes en el país, por tanto, es fundamental, que las soluciones que se obtengan para un problema de asignación de tripulación contemplen las reglas laborales, pues de otro modo la solución no podrá ser aplicada.

El IDVSP *(Integrated Duty and Vehicle Scheduling Problem*) es un modo de resolver los problemas VSP y DSP de forma integrada. Una solución al IDVSP debe contemplar entonces, tanto los objetivos esperados para el VSP como los del DSP, y además, no debe ser fruto de la aplicación sucesiva de la resolución de éstos dos problemas. Los objetivos principales del IDVSP, serán entonces, minimizar la cantidad de kilómetros de viajes expresos realizados por la flota, minimizar la cantidad de vehículos de la flota utilizados, objetivos éstos, relacionados al uso de los vehículos. Y a la vez, minimizar la cantidad total de horas trabajadas, así como la cantidad de jornales de personas requeridas para cubrir dicho trabajo, cumpliendo con todas las restricciones a las cuales se sujeta el problema.

2.2. Resolución de Problemas de Optimización

Para problemas de complejidad *NP* donde se incluyen gran número de problemas de optimización combinatoria, es necesario utilizar algoritmos de resolución más eficientes que una búsqueda exhaustiva, principalmente debido a que el incremento del tamaño del problema impacta de manera súper-polinomial a la complejidad del algoritmo de resolución, lo que lleva a costos, en términos de tiempo computacional, mucho mayores sino imposibles de cumplir en tiempo y forma. Entre los métodos de resolución podemos encontrar técnicas analíticas (aplicables a un conjunto reducido de problemas), métodos exactos que utilizan técnicas de enumeración total o parcial como ser *backtracking*, programación dinámica o *Branch & Bound* o basados en programación matemática (*Simplex*, Punto Interior) que sufren considerablemente el problema de complejidad mencionado. También existen métodos de resolución por aproximación (Métodos Markovianos, Inferencia Probabilística) o métodos aleatorios (Monte Carlo, Las Vegas) que buscan encontrar soluciones con un margen de error establecido de manera de mitigar los costos[MCPBO].

Particularmente y más cercanas al enfoque aquí presentado para la resolución del problema en cuestión, también existen técnicas de algoritmos conocidas como heurísticas o metaheurísticas[METAH] las cuales constan de técnicas basadas en procedimientos conceptualmente simples que encuentran soluciones de buena calidad, no necesariamente óptimas, a problemas complejos procurando hacerlo de un modo

sencillo y eficiente. Existe un amplio espectro de técnicas heurísticas, muchas de ellas basadas en sentido común o emulación de fenómenos bien conocidos. También existen las denominadas heurísticas específicas, las cuales incorporan conocimiento e información del problema de optimización a resolver en beneficio de mejores resultados a menor costo.

2.3. Algoritmos Genéticos

Para introducir Algoritmos Genéticos, debemos definir las metaheurísticas[MIS], ya que los algoritmos genéticos, pertenecen al grupo Algoritmos Evolutivos que se incluyen dentro de las metaheurísticas basadas en población, como se muestra en la Figura 2.6.1. Sobre las metaheurísticas, *podría decirse que son un proceso iterativo de generación que guía a una heurística subordinada, combinando de forma inteligente, diferentes conceptos para la exploración del espacio de búsqueda y usando estrategias de aprendizaje para estructurar la información, con objeto de encontrar, eficientemente, soluciones cercanas al óptimo (Osman y Laporte, 1996).*

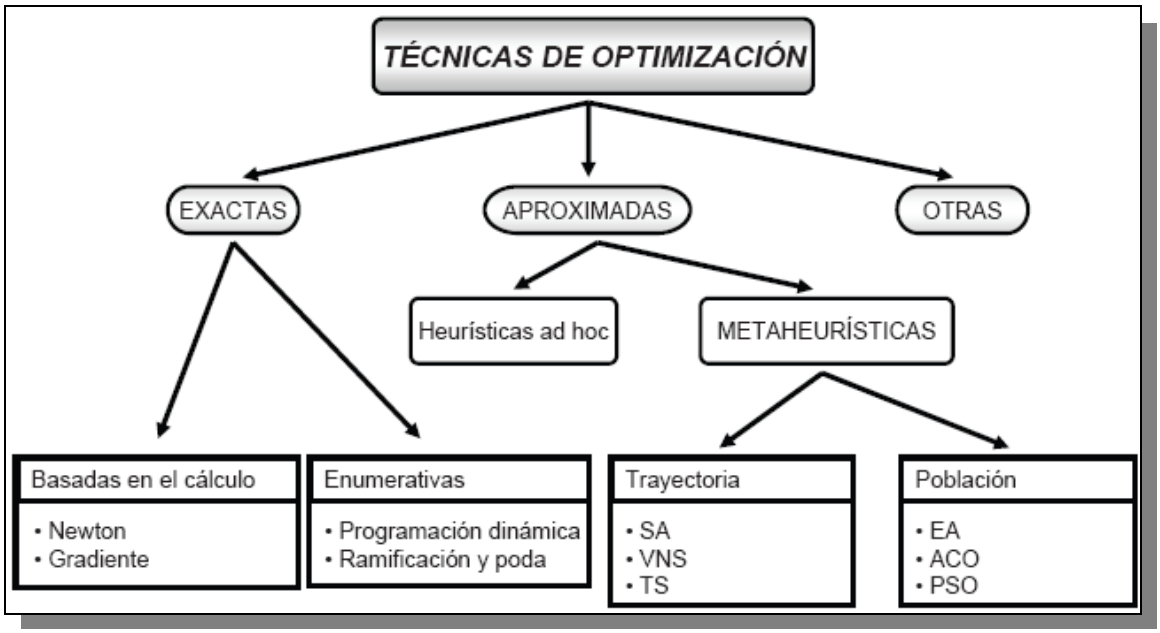

Fig. 2.6.1 Clasificación de técnicas de optimización[MAPRRT].

Es decir, que las metaheurísticas son técnicas de optimización aproximadas, en las cuales se combinan varias técnicas heurísticas de alto nivel, de modo de lograr una exploración del espacio de búsqueda que sea eficiente y eficaz. El término fue mencionado por primera vez por Glover[GLO].

Particularmente existen técnicas de computación evolutiva que forman un conjunto de heurísticas que pueden ser y han sido utilizadas exitosamente para la resolución de una variada gama de problemas tanto en áreas de optimización combinatoria, búsqueda de información, aprendizaje automático, entre otras. Estas técnicas se basan en la emulación de procesos de la evolución natural, muchos de ellos

identificados por Charles Darwin en: *El Origen de las Especies por medio de la Selección Natural: la selección natural, la reproducción y la diversidad genética de los Individuos (1859).*

Estas técnicas de computación evolutiva (Algoritmos Evolutivos) [TEVOL] donde se incluyen los Algoritmos Genéticos (AG) definen mecanismos que utilizan ideas de la evolución natural de los seres vivos para resolver problemas de optimización y búsqueda cuya formulación fue presentada detalladamente en el texto de Goldberg (1989)[GOLD]. Un algoritmo evolutivo trabaja sobre individuos que representan soluciones (potenciales, parciales o completas) codificadas con algún esquema o mecanismo establecido (por ejemplo, vectores, matrices o árboles). A diferencia de otras metaheurísticas, en cada iteración no se opera sobre una única solución (individuo), sino que se trabaja sobre un conjunto de soluciones llamado población (*P)* dada la analogía. La idea se basa en que estos individuos interactuarán entre sí, siguiendo principios darwinianos de la evolución natural con el objetivo de producir iterativamente –en cada generación– mejores soluciones al problema (individuos más aptos). Cada individuo $i \in P$ será evaluado mediante una función de *fitness* o aptitud, que tomando en cuenta la realidad planteada (problema) definirá un valor de modo que cuanto mayor sea el *fitness* de un individuo, más apto es, y mejor es la solución que éste representa.

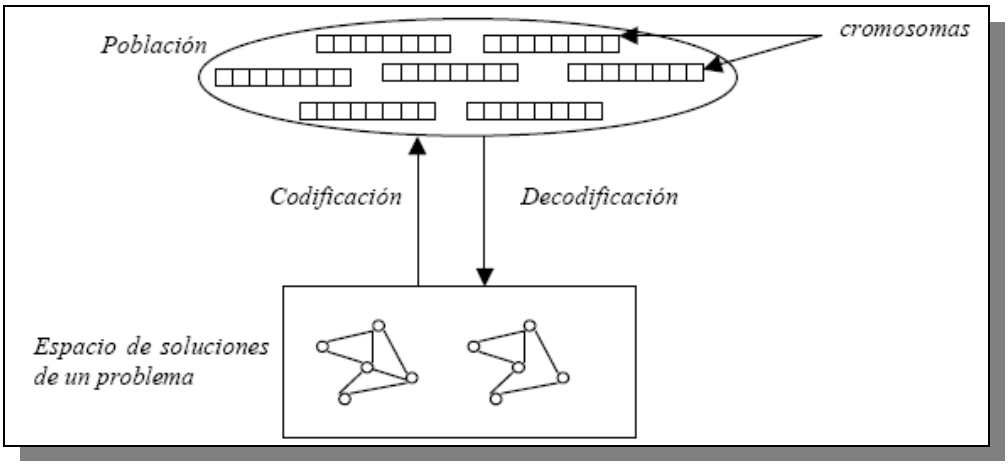

Fig. 2.6.2 Codificación de soluciones en un algoritmo genético.

En cada iteración se aplican un conjunto de operadores evolutivos determinados que combinan y modifican a los individuos de la población. En el esquema clásico se opera en tres etapas: *selección, cruzamiento* y *mutación*. El operador de selección se encarga de elegir algunos individuos de la población que tendrían la posibilidad de reproducirse (usualmente se busca incluir en esta selección a algunos de los individuos con mejores valores del *fitness* – más aptos). En términos computacionales, es el proceso que determina candidatos adecuados, de acuerdo a sus valores de *fitness*, para la aplicación de los operadores evolutivos con el objetivo de engendrar la siguiente generación de individuos. Luego se aplica el operador de cruzamiento, que consta de la combinación de dos individuos (llamados padres - *parents*) de acuerdo a algún mecanismo determinado, para generar dos nuevos individuos llamados hijos (*offsprings)*. Por último, el operador de mutación suele aplicarse con baja probabilidad

para realizar pequeñas modificaciones sobre los individuos buscando aportar diversidad a la población. Los operadores de cruzamiento y mutación trabajan sobre los individuos, particularmente siguiendo la analogía con la evolución natural, sobre lo que se denomina cromosoma, es decir, sobre la codificación de las soluciones. El algoritmo se ejecuta, a lo sumo, una cantidad prefijada de iteraciones, simulando la evolución de las generaciones en el tiempo, y a cada una de las poblaciones obtenidas sucesivamente se la identifica por su número de iteración (generación). Un esquema de este tipo de algoritmos se presenta en la Figura 2.6.3.

```
Inicializar(P(0))
generacion=0
mientras(no CriterioParada) hacer
      Evaluar(P(generacion))
      Padres = Seleccionar(P(generacion))
      Hijos = Aplicar Cruzamiento(Padres)
      Hijos = Aplicar Mutacion(Hijos)
      NuevaPob = Reemplazar(Hijos,P(generacion))
      generacion++
      P(generacion) = NuevaPob
Fin
retornar MejorSolucionEncontrada
```

Fig. 2.6.3 Pseudocódigo de algoritmo genético clásico.

2.4. Antecedentes

2.4.1. Algoritmos Genéticos Aplicados a Optimización

Existen múltiples antecedentes acerca de la aplicación de metaheurísticas, en particular, de algoritmos genéticos, a los problemas del transporte público relativos al ordenamiento de flota y tripulación, obteniendo buenos resultados utilizando algoritmos genéticos acordes a la complejidad y tiempos que el problema implica. Particularmente, el trabajo de Helena R. Lourenço et al [PAIX], plantea un análisis y comparativo de su uso contrastado con otras técnicas como ser con *Tabu Search*, y programación lineal, *ILP*, para una realidad de transporte público.

Si bien no existe unanimidad sobre la aplicabilidad de los algoritmos genéticos, la amplia gama de problemas resueltos exitosamente en diversas áreas han popularizado a esta técnica evolutiva como una de las más versátiles y robustas. Algunos investigadores del área concuerdan en señalar a los algoritmos genéticos como altamente útiles para resolver problemas con espacio de soluciones de dimensión elevada o no muy bien comprendido de antemano, donde el diseño de operadores específicos de *hill-climbing*[HILLC] no sea sencillo, en problemas multimodales donde los métodos heurísticos tradicionales pueden quedar atrapados en óptimos locales o en

casos donde no es estrictamente necesario obtener una solución óptima al problema, sino que una buena solución aproximada sería suficiente.

2.4.2. SDVSP

El problema SDVSP *(Single Depot Vehicle Scheduling Problem),* consiste únicamente del ordenamiento de vehículos en un escenario donde solamente se dispone de un depósito para los vehículos. Este problema ya ha sido resuelto de manera exacta en tiempo polinomial utilizando diversos métodos de modelado del problema[DESAUL].

2.4.3. MDVSP

El problema conocido como MDVSP *(Multiple Depot Vehicle Scheduling Problem),* consiste principalmente en la asignación de una flota de vehículos disponible, la cual podrá encontrarse distribuida en distintos depósitos, a los viajes programados, procurando realizar dicha asignación con el objetivo de minimizar los costos definidos.

El problema MDVSP ha sido definido por Bodin et al[BODIN83] en el año 1983. En el año 1987 Bertossi et al[BERT87], mostraron que se trata de un problema *NP-Hard*. A su vez, existen enfoques que simplifican el mismo, intentando resolverlo como múltiples instancias del SDVSP, hasta el uso de técnicas basadas en *Multicommodity Flow problem, ILP, Branch and Cut[LOBEL]*.

2.4.4. IDVSP

El problema conocido como IDVSP *(Integrated Depot Vehicle Scheduling Problem),* genera la asignación de la flota de vehículos, asignando simultáneamente los jornales de trabajo de la tripulación necesarios para cubrir los viajes.

Este problema, atrajo la atención de los investigadores recién a finales de los noventa. Un ejemplo de esto, es el enfoque planteado por Freling et al[MAIVCS] en el cual se ataca el problema IDVSP considerando un único depósito para los vehículos. La versión multi-depósito de este problema, ha sido abordada por Huisman et al[MDIVCS].

Ambos enfoques, han demostrado tener mejores resultados que el enfoque secuencial, en el cual se asignan primero los vehículos y luego la tripulación [DESAUL].

En este trabajo se buscará abordar el problema de manera integrada para múltiples depósitos.

2.5. Especificación del Problema IDVSP

Como se ha mencionado previamente, el IDVSP es el problema en el cual se debe obtener una asignación óptima de tripulación y flota. A continuación se

especifican los datos de entrada que se requieren para resolver dicho problema. La solución, o sea el dato de salida, es un libro de servicios, que consiste en una tabla que describe los jornales a realizar incluyendo, para cada jornal, todos los eventos del mismo (viajes activos, viajes expresos, descansos).

Dada la realidad del problema, el mismo puede especificarse de la siguiente manera: Dado un conjunto $T = \{1, 2,..., n\}$ de viajes en la tabla de viajes (*timetable*), donde el viaje *i* comienza a la hora s_i y llega a destino a la hora e_i, denotamos como t_{ij} el tiempo que toma en realizar dicho viaje. Diremos que dos viajes son compatibles, si y solo si pueden ser realizados consecutivamente por un mismo vehículo, esto es, *sii* $e_i + t_{ij} \leq s_j$, *entonces i es compatible con j*.

Sea $K = \{1,2,\ldots,m\}$ el conjunto de *m* depósitos, dónde se guardan los vehículos que deberán realizar los viajes, decimos que el depósito *k* alberga v_k vehículos donde para todo vehículo asignado que efectúe un *pull out trip* deberá realizar el correspondiente *pull in trip* al final de su jornada.

Como se mencionó, a diferencia de los viajes activos, los cuales no generan costo, sí se incurre en un costo cada vez que un vehículo realiza tanto un *deadhead, pull out* o *pull in trip*, donde denotaremos como c_{ij} al costo que implica ir de *i* a *j*. Por lo que el costo de un servicio o s*chedule*, de un vehículo determinado será, simplemente la suma de los costos de los viajes que realiza (contando únicamente los costos por asignación de flota).

Dicho lo anterior, el problema, considerado solamente como un problema de asignación de flota, puede ser definido como el problema de encontrar un conjunto de *schedules* para los vehículos, tales que: cada viaje activo sea cubierto por exactamente un *schedule*, como máximo se podrán asignar v_k *schedules* para cada depósito *k* y la suma de los costos de los *schedules* sea mínima.

La especificación formal del VSP[DESAUL] se presenta a continuación. Cabe aclarar que esta formulación corresponde a la aplicación de una red de flujo *multicommodity*, en la cual se asocia un *commodity* a los vehículos que operan en un mismo depósito, de modo que la formulación asegura que cada vehículo sale y retorna siempre al mismo, en dicha formulación a cada depósito se le asocia un grafo $G^k = (N^k, A^k)$ donde N^k, el conjunto de nodos se define como $N^k = T \cup \{k\}$, el conjunto de arcos A^k, se define como $A^k = C \cup (\{k\} \times T) \cup (T \times \{k\})$, donde *C* es un subconjunto de *T x T*, que contiene un arco $(i, j) \in T \times T$ sii los viajes *i* y *j*, son compatibles. Adicionalmente, se define una variable binaria X_{ij}^k, para cada depósito *k*, y cada arco *(i, j)* que indica el flujo o no de buses originados en el depósito *k* sobre el arco *(i, j)*. Las restricciones presentadas en la formulación del VSP (1 al 4), se explican luego cuando se define el problema IDVSP.

$$\textit{Minimize} \qquad \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij} \, X_{ij}^k$$

subject to:

$$\sum_{k \in K} \sum_{i:(i,j) \in A^k} X_{ij}^k = 1, \qquad \forall \, j \in \mathcal{T} \tag{1}$$

$$\sum_{j \in \mathcal{T}} X_{k,j}^k \leq v^k, \qquad \forall \, k \in K \tag{2}$$

$$\sum_{i:(i,j) \in A^k} X_{ij}^k - \sum_{i:(j,i) \in A^k} X_{ji}^k = 0, \qquad \forall \, k \in K, \; j \in \mathcal{T} \cup \{k\} \tag{3}$$

$$X_{ij}^k \in \{0,1\}, \qquad \forall \, k \in K, \; (i,j) \in A^k. \tag{4}$$

La tripulación también debe ser considerada en los costos. Para esto, definiremos los puntos de relevo, los cuales representan lugares, entiéndase ciudades o depósitos, en los cuales puede ocurrir un relevo de tripulación. Dada una agenda para un vehículo, el mismo podría pasar por varios puntos que no sean puntos de relevo. Esos trayectos que no incluyen puntos de relevo se denominan segmentos. Definimos entonces el conjunto *S* de segmentos, los cuales son bloques de viajes que deben ser cubiertos de continuo por un solo conjunto de tripulantes (esto se debe a que ninguno de los viajes del segmento pasa por un punto de relevo). Adicionalmente, definimos a D, como el conjunto de rutinas (*duties*), actividades realizadas por la tripulación, válidas y denotamos como c_d el costo de realizar la rutina d y a^s_d un parámetro binario que toma el valor 1 si la rutina d cubre el segmento *s* y 0 en otro caso. Finalmente, definimos la variable Y_d para cada rutina $d \in D$ que indica si la rutina misma se realiza en la solución (i.e. si se encuentra asignada). Las restricciones presentadas en esta formulación, se explican luego en la definición del problema IDVSP (Restricciones 5 y 7).

Entonces, el problema DSP, puede formularse de la siguiente manera:

$$\textit{Minimize} \qquad \sum_{d \in D} c_d \, Y_d$$

subject to:

$$\sum_{d \in D} a_d^s Y_d = 1, \qquad \forall \, s \in S \tag{1}$$

$$Y_d \in \{0,1\}, \qquad \forall \, d \in D. \tag{2}$$

Finalmente, el problema IDVSP, puede formularse como sigue:

$$\textit{Minimize} \qquad \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}\, X_{ij}^k + \sum_{k \in K} \sum_{d \in D^k} c_d Y_d^k$$

subject to:

$$\sum_{k \in K} \sum_{i:(i,j) \in A^k} X_{ij}^k = 1, \qquad \forall\, j \in \mathcal{T} \tag{1}$$

$$\sum_{j \in \mathcal{T}} X_{k,j}^k \leq v^k, \qquad \forall\, k \in K \tag{2}$$

$$\sum_{i:(i,j) \in A^k} X_{ij}^k - \sum_{i:(j,i) \in A^k} X_{ji}^k = 0, \qquad \forall\, k \in K,\ j \in \mathcal{T} \cup \{k\} \tag{3}$$

$$X_{ij}^k \in \{0,1\}, \qquad \forall\, k \in K,\ (i,j) \in A^k \tag{4}$$

$$\sum_{k \in K} \sum_{d \in D^k} a_d^s Y_d^k = 1, \qquad \forall\, s \in S \tag{5}$$

$$\sum_{d \in D^k} b_{dij} Y_d^k - X_{ij}^k = 0, \qquad \forall\, k \in K,\ (i,j) \in A^k \tag{6}$$

$$Y_d^k \in \{0,1\}, \qquad \forall\, k \in K,\ d \in D^k. \tag{7}$$

El significado de las restricciones es el siguiente: la (1) asegura que cada viaje es realizado exactamente una vez, la (2) limita el número de vehículos disponibles por depósito, la restricción (3) es una restricción de conservaciones de flujo en el grafo, y la (4) restringe la variable X a valores binarios. La restricción (5) asegura que cada segmento es cubierto por un rutina, la (6) establece la relación entre el problema de asignación de flota y de tripulantes, que establece que cada viaje inactivo expreso es cubierto por un chofer. Finalmente, la restricción (7) limita a valores binarios la variable Y. A su vez, la función objetivo minimiza la suma de costos de trabajo y vehículos.

Luego de formulado el problema de programación lineal, el mismo podría resolverse utilizando diversos métodos, desde relajaciones, generación de columnas, u otros métodos, exactos o no. En este punto, se destaca que los métodos aproximados, han tenido mejores resultados en cuanto a tiempo de ejecución en instancias de gran cantidad de viajes[BDSUTS].

2.5.1. Datos de Entrada

2.5.1.1. Tabla de distancias entre ciudades y depósitos

Dado un conjunto de ciudades que incluya los puntos de inicio y fin de todos los viajes, se deberá proveer, en el formato adecuado, una tabla que contenga la distancia entre cualquier par de ciudades. Esta información es utilizada para el cálculo de distancias recorridas por los viajes, tanto activos como expresos. La Figura 2.8.1.1 es un ejemplo de la tabla mencionada. En dicha tabla, además, se deberá especificar la distancia entre las ciudades y aquellos depósitos que no pertenezcan a ninguna de ellas de modo de poder determinar los costos de los viajes *pull in* y *pull out* para esos casos.

	Aceguá	Artigas	Atlántida	Bella Unión	Canelones	Cardona	Carmelo	Colonia	Chuy
Aceguá	*	452	417	587	437	584	625	566	318
Artigas	452	*	650	135	555	581	540	611	651
Atlántida	417	650	*	663	67	217	281	222	294
Bella Unión	587	135	663	*	593	487	496	567	794
Canelones	437	555	67	593	*	141	210	151	351
Cardona	584	581	217	487	141	*	106	93	498
Carmelo	625	540	281	496	210	106	*	76	554
Colonia	566	611	222	567	151	93	76	*	491
Chuy	318	651	294	794	351	498	554	491	*

Fig. 2.8.1.1 Ejemplo de tabla de distancias para algunas ciudades del Uruguay.

2.5.1.2. Tabla de viajes a realizar

La tabla de viajes, consiste en la especificación del conjunto de viajes a cubrir. Los datos de cada viaje deben ser: ciudad de origen, ciudad de destino, horario de partida, horario de llegada y un conjunto de categorías válidas de vehículos que podrán cubrir dicho viaje. Usualmente las categorías se utilizan para discriminar los vehículos de una flota heterogénea entre distintos tipos de capacidades, comodidades, modelos de vehículos, etc. Un ejemplo de esto, se puede visualizar en la Figura 2.8.1.2, la cual consiste de una programación de seis viajes. Uno de los conceptos más importante que se definirá a partir de esta tabla es el de compatibilidad entre dos viajes cualesquiera.

Cada viaje debe tener asociado, un conjunto ordenado de categorías de vehículos aceptables (aptos) para realizarlo. Este conjunto será utilizado como una lista de preferencias, la misma determinará cuáles categorías de vehículos son factibles de utilización para la ejecución de un viaje, cuyo orden de preferencia estará definido por

la empresa. El sistema intentará asignar los vehículos cuya categoría tenga mayor valor en la escala de preferencias, definida por el orden en el cual están especificadas las categorías (prioridad descendente), y nunca asignará un vehículo que no sea factible, o sea, que no esté presente en la lista de categorías aceptables.

	Origen	Destino	Inicio	Fin	Categorías
1	Atlántida	Canelones	13:00	14:00	3, 2
2	Artigas	Bella Unión	13:00	15:00	1
3	Cardona	Carmelo	13:00	14:30	1, 2
4	Canelones	Chuy	14:30	19:00	1, 3
5	Carmelo	Colonia	15:00	16:00	2, 1
6	Colonia	Cardona	17:00	18:30	1, 3, 2

Fig. 2.8.1.2. Ejemplo de tabla de viajes (activos) a cubrir.

En el caso de la Figura 2.8.1.2 se puede encontrar que el viaje 1 es compatible con el 4, esto se debe a que el viaje 1 llega a Canelones a las 14:00, y el viaje 4 parte de Canelones a las 14:30. De la misma figura, por ejemplo, se observa que el viaje 1 no es compatible con el 2, esto ocurre ya que el viaje 1 termina en Canelones a las 14:00, y debería poder llegar a Artigas a las 13:00. La compatibilidad entre los viajes 1 y 5 queda sujeta a la factibilidad de realización en una hora o menos de un viaje expreso entre Canelones y Carmelo. Para esto, se utiliza la tabla de tiempos para viajes expresos definida en la sección 2.5.1.3.

2.5.1.3. Tabla de tiempos para viajes expresos

Los viajes expresos, normalmente se llevarán a cabo con una duración menor, usualmente a velocidad más alta que los viajes activos, ya que no transportan pasajeros. Es así, que se proveerá una tabla, conteniendo, para cada par origen-destino, la duración de un viaje expreso entre ellos. Dicha tabla será análoga a la de la Figura 2.8.1.2 excluyendo la columna de categorías y en vez de contener el dato de las distancias entre ciudades, contendrá la duración de viajes expresos entre las mismas.

2.5.1.4. Cantidad y categoría de vehículos disponibles por depósito

Por último, dada la existencia de múltiples depósitos se requerirá conocer la distribución de la flota entre estos, es decir la cantidad de vehículos disponibles en cada uno de los depósitos. Cada vehículo pertenecerá a una categoría determinada, establecida por su capacidad, comodidades, etc., la cual será considerada a la hora de asignarle viajes a realizar.

2.5.1.5. Puntos de Relevo

Para el conjunto de ciudades involucradas (que serán origen y destino de los viajes y depósitos), se debe especificar si la misma es un punto autorizado para relevo de personal. Esto implica la posibilidad de poder realizar los relevos de tripulación correspondientes a distintos turnos, lo cual representa una restricción de la factibilidad para las soluciones generadas.

2.5.1.6. Reglas Laborales

Si bien hay una sección dedicada enteramente a las reglas laborales, corresponde mencionar que también se deben proveer como dato de entrada cuáles son las reglas laborales que deben cumplirse, así como valores o constantes referentes a las mismas. Para la especificación de ellas se provee al usuario de un lenguaje definido, el cual se detalla en la sección 4, pero cabe mencionar que las reglas podrán establecer restricciones sobre cualquiera de los conceptos modelados, los cuales serán tenidos en cuenta para la validación de cada libro de servicio. A modo de ejemplo las reglas podrán establecer restricciones sobre propiedades de los turnos. Por ejemplo, condicionar la duración de lo mismos, kilometraje a recorrer, relaciones entre ellos, e inclusive sobre las actividades realizadas durante los mismos, como ser descansos o viajes.

Para la gestión de estas funcionalidades se implementa el Módulo de Reglas Laborales (WRM) el cual es presentado en el anexo C.

2.5.2. Salida – Libro de Servicios

La salida especificará el libro de servicios a realizar en un formato definido. Cada servicio será realizado por un único vehículo y estará compuesto por uno o más turnos, correspondientes a una jornada laboral de un chofer o tripulación. Análogamente, para cada turno se especifican los eventos que lo componen y sus detalles dependiendo del tipo, donde se incluyen por defecto, tres tipos básicos de eventos: viajes activos, viajes expresos y descansos.

El formato de salida (obviando datos estadísticos de la ejecución y solución) consta de un listado con los datos mencionados previamente de la manera que indica la Figura 2.8.2.1. También se provee una salida compatible con la interfaz gráfica (FingLab) desarrollada por el Grupo de Investigación Operativa (IO)[UIIO] que permite la visualización del libro a través de Diagramas de Gantt.

```
<Datos de Turno>
<Datos de Vehiculo>
<Datos de Listado de Eventos>
      <Datos de Evento>
```

Fig. 2.8.2.1. Formato de especificación de turno.

```
<Turno>
. . .
Turno: 1
Vehiculo: 1 Deposito: 0
Listado de Eventos: (10, 395mins)
      Activo            1     0     13:30 14:25
      Descanso                      14:25 14:40 (15mins)
      Expreso           0     2     14:50 15:30
      Activo            2     0     15:30 16:10
      Descanso                      16:10 16:25 (15mins)
      Activo            0     2     16:50 17:30
      Expreso           2     0     17:50 18:30
      Activo            0     2     18:30 19:10
      Expreso           2     0     19:10 19:50
      Descanso                      19:50 20:05 (15mins)
. . .
<Turno>
```

Fig. 2.8.2.2. Ejemplo de formato de especificación de turno. Para cada evento se indica tipo, origen, destino, horario de inicio y fin.

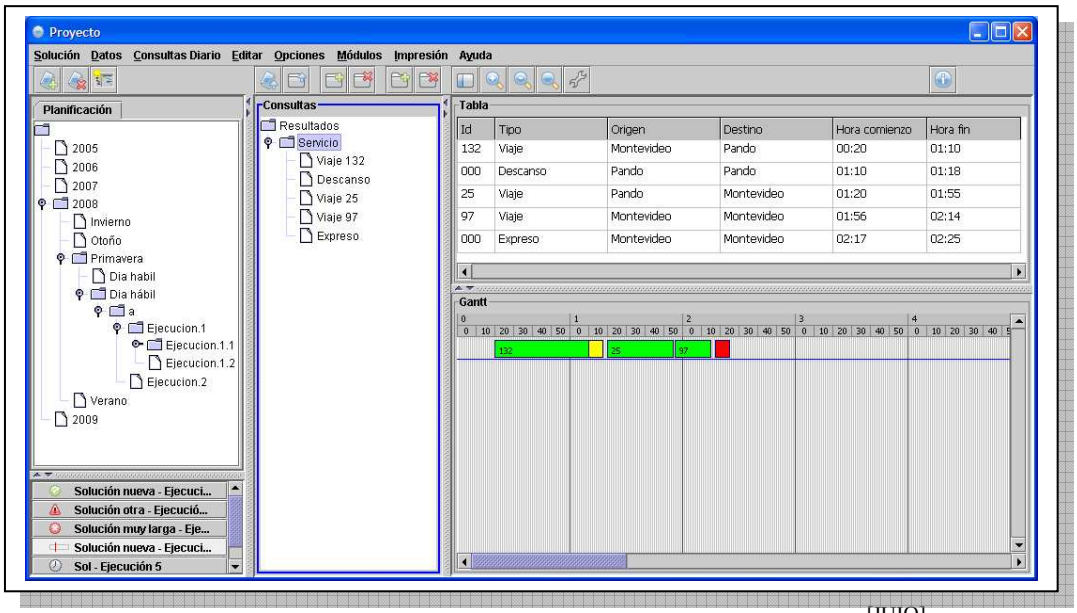

Fig. 2.8.2.3. Interfaz gráfica para la visualización de libros de servicio. [IUIO]

3. Algoritmo Genético

3.1. Representación de la solución

Como se mencionó previamente, en la sección 2.6, los algoritmos genéticos no trabajan directamente sobre la solución, sino que lo hacen sobre una representación de la misma, usualmente denominadas cromosomas (por su analogía con la evolución natural biológica). Cabe notar que este mecanismo tenderá a simplificar la aplicación de operadores evolutivos ya que facilita la analogía con el proceso natural. Para ello se debe definir una función de codificación (c*ode()*) sobre los puntos del espacio de soluciones, que mapee de manera unívoca todo punto del espacio de soluciones a una codificación del cromosoma.

$$code : Espacio\ de\ Soluciones \rightarrow cromosoma$$

Dicho lo anterior, cabe tener en cuenta la importancia de los mecanismos de codificación de los individuos en el proceso de búsqueda de estos algoritmos, y el impacto que puedan tener los mismos sobre él (por ejemplo en materia de costos de ejecución).

Considerando lo anterior y habiendo evaluado diversas posibilidades para la codificación y representación de las soluciones, se presenta a continuación la opción seleccionada. El factor decisivo para dicha selección, teniendo en cuenta la complejidad del problema, se basa en la simplicidad que la misma ofrece, lo que implica rápido acceso a sus valores, bajo costo de almacenamiento y serialización (codificación/decodificación) de soluciones. Esta representación facilita la validación de la completitud de las soluciones en cuanto al cubrimiento de los viajes a realizar, cuyo costo es considerable y dependiente de la complejidad de las restricciones. A su vez, previene resultados y operaciones con poblaciones o individuos no factibles (viajes activos no asignados o asignaciones duplicadas). Emplear una codificación simple, que represente la abstracción de una realidad compleja puede descartar propiedades particulares de cada individuo y provocar pérdida de diversidad de la población. Por esto, se pretende que dicha pérdida sea mitigada mediante su consideración en los distintos operadores y técnicas auxiliares implementadas en el algoritmo, como por ejemplo la de *ajuste dinámico de la probabilidad de mutación* presentada en la sección 3.5.

La representación de la solución (cromosoma) consistirá principalmente de una matriz (*M)* de dimensión: *2 x cantidad_de_viajes_activos*. La misma puede ser interpretada como dos vectores donde el primero (*vehiculos[])* representará la asignación de la flota a los viajes activos, en el cual *vehiculos[i]* indica, para cada viaje *i*, el identificador del vehículo que lo realizará. Análogamente, el segundo componente de *M* será un vector (*turnos[])* que representará la asignación de turnos, donde *turno[i]* indica el identificador del turno al que pertenece el viaje *i*. Cada turno corresponderá a un chofer (o un conjunto de tripulantes), para el cual luego se generará el conjunto de eventos a realizar (incluyendo viajes activos, expresos, descansos, etc.).

T1	V1	A
T2	V2	C
T3	V1	A
T4	V3	E
T5	V3	E
T6	V2	D
T7	V1	B

En la solución anterior donde la realidad presenta un total de siete [T1..T7] viajes a cubrir (viajes activos - *active trips*), los viajes que realiza el vehículo 1, son cubiertos, en parte por el turno A, y en parte por el turno B, los viajes que realiza el vehículo 2, son realizados por los turnos C y D, y los viajes que realiza el vehículo 3, son cubiertos únicamente por el turno E.

La construcción de cada solución a través de la codificación mencionada (cromosoma), es decir, la generación del libro de servicios con sus correspondientes turnos y eventos a partir de los vectores de asignación, se compone de 4 pasos principales. Se recorre el vector de *vehiculos[]* teniendo en cuenta que cada vehículo representará un servicio. Para cada vehículo (servicio) asignado a un viaje, se construyen los turnos identificados para cada uno de sus viajes. Dentro de estos turnos se determinan e incluyen los viajes activos asignados y los viajes expresos necesarios teniendo en cuenta el destino y origen de viajes consecutivos. Finalmente para cada turno se asignan (acorde al escenario) los descansos según las reglas laborales especificadas o demás eventos que sean definidos. La decodificación de cada uno de los eventos que componen la solución es determinista.

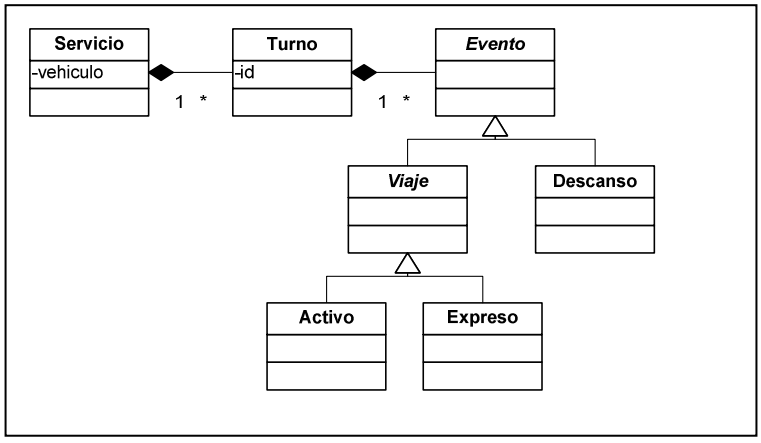

Fig. 3.1 Composición de un servicio (Diagrama de Clases UML)

3.2. Población Inicial

Previo a la generación de soluciones iniciales, el sistema genera una tabla de compatibilidades, que indica, para todo par de viajes, si los mismos son compatibles o no. Esta generación previa se debe a que dicha tabla, se consultará permanentemente a lo largo de la ejecución del algoritmo, a fin de verificar la factibilidad de las soluciones generadas en cada paso.

Para obtener una población inicial, se han decidido desarrollar cuatro modos de inicialización diferentes. Todos basados en la idea de construir poblaciones de individuos factibles y completos, es decir, que cubran todos los viajes activos cumpliendo con todas las restricciones establecidas.

Cada solución inicial generada, es procesada con dos objetivos. Por un lado verificar el cumplimiento de restricciones y reglas definidas sobre los viajes y vehículos, y en segundo lugar, teniendo el anterior como precondición, la generación de los turnos (asignación de tripulación) que fueren necesarios para cumplir con la asignación de flota realizada. Si la solución no fuera factible desde el punto de vista de asignación de turnos de personal, se intentará generar otra solución.

3.2.1. Inicialización Secuencial

Esta técnica intenta minimizar la cantidad de vehículos utilizados, para esto, dado un viaje, intenta reutilizar un vehículo que ya esté en uso (haya sido asignado). Esta estrategia, responde a la idea intuitiva de intentar asignar viajes compatibles de manera secuencial a un mismo vehículo, de modo de reducir el kilometraje expreso recorrido (*deadhead, pull-in, pull-out trips*) y tiempos de espera correspondientes al tiempo en que un vehículo y su tripulación se encuentran a la espera de la salida de su próximo viaje activo asignado. Cabe notar que el tiempo de espera, también llamado tiempo muerto, no es tenido en cuenta como descanso para la tripulación por lo que implica un gasto en recursos ociosos que debería minimizarse.

En el siguiente pseudocódigo, se asume que la función *noAsignado* retorna *true* cuándo el viaje aún no ha sido asignado y que el método *setVehiculo* controla la factibilidad de la solución, verificando todas las restricciones ya mencionadas en cuanto a compatibilidad entre viajes asignados a un vehículo.

```
for each vehiculo in Vehicles do
          for each viaje in Timetable
                if (viaje.noAsignado())
                      viaje.setVehiculo(vehiculo);
          end;
end;
```

Figura 3.2.1. Pseudocódigo de Inicialización Secuencial.

3.2.2. Inicialización Aleatoria

Esta modalidad genera individuos de manera aleatoria, para esto, recorre la lista de viajes, y asigna un vehículo a cada uno de los viajes de forma aleatoria. En cada paso, verifica que la asignación de ese vehículo, mantenga la factibilidad de la solución respecto a las restricciones definidas. El objetivo del uso de esta técnica, es la obtención de una población inicial diversa, que permita explorar ampliamente el espacio de soluciones de modo que cada individuo pueda aportar lo mejor de sí a lo largo de la evolución del algoritmo con el fin de optimizar soluciones. En la Figura 3.2.2 se presenta un pseudocódigo de la inicialización, al igual que en la sección 3.2.1, el mismo asume que el método *setVehiculo* chequea la factibilidad de la asignación,

```
for each viaje in Timetable do
            viaje.setVehiculo(random(vehiculos));
end;
```

Figura 3.2.2 Pseudocódigo de Inicialización Aleatoria

3.2.3. Inicialización por Entronques

Dada la naturaleza del negocio de las empresas de transporte, es muy común que en la tabla de viajes (*trip table)* aparezcan viajes en ambos sentidos entre dos ciudades, es decir, si una empresa maneja viajes desde una ciudad A hacia la ciudad B, lo más común es que la misma empresa ofrezca los viajes de retorno (viajes desde B hacia A).

El tercer método de inicialización, el método por entronques, es un método utilizado en empresas de transporte basado en lo anterior, dicho método, utiliza los viajes de ida y vuelta entre dos puntos e intenta cumplimentarlos con un mismo vehículo. Éste método tiene una virtud fundamental: el mismo vehículo haciendo viajes de ida y vuelta, tiene grandes probabilidades de salir desde la ciudad de su depósito y retornar a la misma al fin de la jornada sin tener que realizar viajes expresos de *pull-in* o *pull-out*. Además se mantiene la virtud mencionada en el método secuencial, ya que en éste método también se intenta reasignar un mismo vehículo. De ésta manera, podríamos decir que el método de entronques, es un método que intenta minimizar el uso de flota, y a la vez hacer una asignación inteligente con el fin de reducir al mínimo los kilómetros expresos.

En el seudo código presentado en la Figura 3.2.3 se asume que la función *noAsignadoInverso* retorna *true* si un viaje no tiene un coche asignado y además es un viaje de retorno del viaje que se le pasa como parámetro.

```
for each vehiculo in Vehicles do
  for each viaje in Timetable
        if (viaje.noAsignado())
              viaje.setVehiculo(vehiculo);
              ultV = viaje;
              for each viaje in timeTable do
                    if(viaje.noAsignadoInverso(ultV))
                          viaje.setVehiculo(vehiculo);
                          ultV = viaje;
                    end;
              end;
        end;
  end;
end;
```

Figura 3.2.3. Pseudocódigo de Inicialización por Entronques.

3.2.4. Inicialización Combinada

Este método de inicialización combina las técnicas presentadas anteriormente generando cada individuo mediante la aplicación de una de estas, la cual se selecciona en base a un valor probabilístico dado. De este modo se aumenta la diversidad de la población inicial.

3.3. Operadores Genéticos

En esta sección, se presentan las diferentes opciones que hemos utilizado para la implementación de los operadores, dada la representación de las soluciones, y el problema que estamos resolviendo. A continuación, se detallan dichos operadores.

3.3.1. Selección

El operador de selección es uno para los cuales realizaremos pruebas de configuración. El correcto comportamiento de este operador es indispensable para no provocar convergencia prematura, posiblemente causada por individuos mucho mejores que el resto, donde aún encontrándose lejos del óptimo e impulsado por el mecanismo de selección, pueden dominar rápidamente la población (eventualmente sin poder escapar de mínimos locales). Para prevenir esto y conservar la diversidad de la población se propone la utilización de dos métodos distintos. Se propone realizar evaluaciones comparables con los operadores *Rueda de Ruleta* y *Selección Estocástica Universal*, cuyos procedimientos y características se detallan en la sección 3.6.2.

3.3.2. Cruzamiento

Las soluciones de nuestro problema, como vimos, son complejas, y encierran en sí mismas el cumplimiento de varias restricciones de factibilidad que pueden ser verificables a partir de una solución particular, los datos de compatibilidad entre viajes, los puntos de relevo y las reglas laborales.

En el cruzamiento, vamos a intentar entonces, manejar un esquema que permita mantener poblaciones de individuos factibles a través de todas las generaciones.

Para ello, consideramos dos técnicas de cruzamiento, que se describen a continuación, donde, al igual que en la inicialización, contaremos con un parámetro que permitirá determinar la probabilidad de uso de cada una de las técnicas.

La primera técnica es el cruzamiento en un punto. Un ejemplo de cruzamiento de un punto se muestra en la Figura 3.3.2.1, sobre la misma debemos realizar algunas consideraciones, ya que no todas las restricciones sobre las soluciones se siguen cumpliendo luego del cruzamiento.

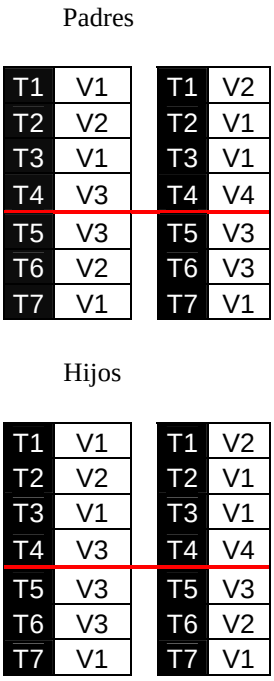

Fig. 3.3.2.1. Ejemplo de cruzamiento de dos soluciones utilizando el operador de cruzamiento a partir de un punto determinado, (posición T5).

Fundamentalmente, la restricción que hay que continuar verificando luego del cruzamiento, es la de compatibilidad de los viajes. Por ejemplo, en el caso de la Figura 3.3.2.1, si el viaje 4 no fuera compatible con el 6, tendríamos que el primer hijo no es factible. Para esos casos, y a modo de fomentar la evolución de las poblaciones, se propone el reintento de dicho cruzamiento, seleccionando otro punto de corte. A fin de

minimizar el impacto en el costo de ejecución de dicha estrategia, se restringe la cantidad de reintentos, mediante la utilización de un parámetro, que representa el máximo de reintentos a realizar.

La segunda técnica planteada para cruzar soluciones es utilizar un operador particular para nuestro problema, basado en los conceptos de *path relinking*[PATHREL].

Este operador, funciona de la siguiente manera, dados dos padres, se sortea una posición del vector solución, y se intercambian los vehículos de esa posición, manteniendo la compatibilidad. Como vemos, ese intercambio, genera dos soluciones, que están "en el camino" entre las dos soluciones originales.

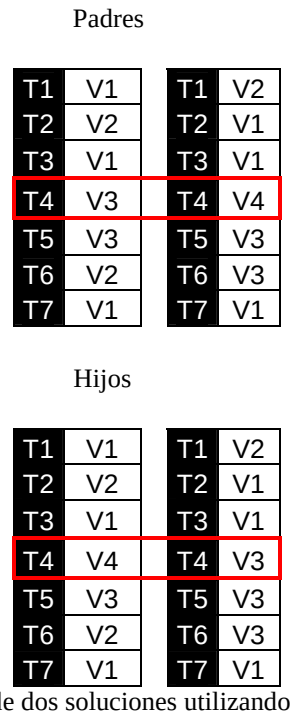

Fig. 3.3.2.2. Ejemplo de cruzamiento de dos soluciones utilizando un operador de cruzamiento basado en *path relinking*. Aquí, se intercambian los vehículos del viaje T4.

En la Figura 3.3.2.2 se ilustra el funcionamiento de este operador. Como vemos, el primer hijo (izquierda) puede ser interpretado como una aproximación de la primer solución (padre izquierdo) a la segunda (padre derecho), ya que ambos pasan a compartir el mismo valor (vehículo asignado) para un mismo viaje luego de la aplicación del operador.

Luego del cruzamiento, dado que las modificaciones que realiza el operador sobre las soluciones no garantiza la factibilidad de los nuevos individuos, es necesario procesarlos a modo de validar el cumplimiento de restricciones laborales y generar los turnos en caso que sean factibles.

Como se mencionó previamente, este operador podría generar asignaciones no factibles, sin embargo, nuestra propuesta es la de mantener la factibilidad en la población. Esto se lleva a cabo, cambiando la posición seleccionada, en el caso de que no resulte factible el intercambio en la posición seleccionada. Nuevamente, se utiliza un

parámetro, que limita la cantidad de reintentos a realizar.

3.3.3. Mutación

La mutación, es un operador que utilizaremos con el fin de aportar diversidad, en términos de exploración del espacio de búsqueda, intentando mediante dicha exploración, no caer en óptimos locales que podrían existir en el espacio de soluciones. Con ese fin, proponemos un operador para el cual, dado un conjunto de posiciones del vector solución, cambie el vehículo utilizado en el viaje, seleccionando aleatoriamente uno de los vehículos disponibles. Para clarificar el ejemplo del funcionamiento de este operador, se incluye la Figura 3.3.3, en la cual se muta la fila correspondiente al viaje 4 (T4), cambiando la utilización del vehículo 3 (V3), por el vehículo 1 (V1). Al operador de mutación queda entonces asociado un parámetro que llamamos porcentaje de mutación, dicho parámetro indica qué proporción de la solución (genes – cantidad de viajes) se quiere mutar, por ejemplo, si se tuviese una solución con 20 viajes y se define el porcentaje de mutación en 10%, los viajes a mutar en cada individuo serán 2.

La decisión de agregar el parámetro de porcentaje de mutación surgió durante el desarrollo del proyecto, debido a que anteriormente se mutaba una única posición, y se observa que durante las ejecuciones al aumentar o disminuir la probabilidad de mutación no cambian significativamente los resultados, ya que el operador, aplicado a problemas de muchos viajes, tiene una influencia muy débil.

Fig. 3.3.3. Ejemplo de mutación de una solución, se cambia la fila correspondiente al viaje 4, cambiando la utilización del vehículo 3, por el vehículo 1.

Luego de la mutación, se procede a invocar al módulo de reglas laborales para generar los turnos correspondientes a la nueva solución.

3.4. Función de *Fitness*

La función de *fitness*, para una solución particular, expresará cuan ajustada, o cuan buena, es una solución para nuestro problema.

En éste caso, como se mencionó anteriormente, los intereses que determinan cuando una solución será mejor, pueden variar e incluso depender de la relación entre ellos. Su influencia en el mecanismo del algoritmo es importante ya que pese a que

actúa como una caja negra para el proceso evolutivo, la función de *fitness* guía el mecanismo de exploración, al actuar representando al entorno que evalúa la aptitud de un individuo solución para la resolución del problema[GOLD].

En principio y teniendo en cuenta uno de los objetivos principales ya planteados, podría considerarse como función objetivo la de minimización de kilómetros expresos. Lo que significaría que una solución será mejor que otra, cuantos menos kilómetros de viajes expresos se deban realizar para llevarla a cabo, incluyendo los viajes de *deadhead, pull-out* y *pull-in*.

Informalmente, esto se reduce, a sumar la cantidad de kilómetros expresos que se deben realizar para cumplir los viajes asignados a cada vehículo. Cuanto más grande es este valor, menos conveniente es la solución, por lo tanto, se podría usar el inverso de esa función como función de *fitness*. A pesar de que dicho planteo facilitaría enormemente la tarea de optimización, como ya mencionamos ese objetivo representa tan solo uno de los intereses posibles, donde en particular y dada su naturaleza debería ser considerado, pero no tendría porqué ser considerado como determinante independientemente del contexto o realidad en que se maneje.

En particular, en esta oportunidad se plantean objetivos adicionales como lo son la minimización de vehículos asignados, minimización de horas de trabajo a realizar, minimización de turnos. Es así, que dados los distintos objetivos y funciones planteadas a optimizar, surge el problema de la necesidad de evaluar múltiples funciones objetivo probablemente con codominios muy distintos no solo en materia de cantidad sino de significado y unidades. Para ello y basados en el esquema planteado por Paixao[PAIX] proponemos la utilización de una función multiobjetivo *z(x)*, donde el problema de combinar los distintos intereses o funciones objetivo podrá ser visto como minimizar dicha función, como se plantea a continuación:

$$\text{Min} \quad z(x) = (f_1(x), f_2(x), \ldots, f_K(x))$$

Nuevamente, dicha función deberá resolver problemas de escala de valores como por ejemplo, si una función objetivo $f_1(x)$ produce resultados entre 0 y 3.000.000 y otra $f_2(x)$ produce resultados entre 1 y 5; entonces, la definición de la función $z(x)$ deberá equilibrar, transformar o escalar los valores de las funciones de manera representativa con respecto a los objetivos e intereses propuestos.

Las funciones propuestas para nuestro caso son las siguientes: La cantidad de kilómetros expresos, la cantidad de vehículos utilizados, la cantidad de jornales, la cantidad de horas totales y agregamos una penalización por turnos que no alcancen una duración mínima (por ejemplo, podría ser de interés penalizar soluciones con turnos de 2 horas, cuando el turno normal fuera de 8 horas).

A continuación, se especifican formalmente las funciones, en primer término, la cantidad de kilómetros expresos se expresa utilizando el conjunto *V* de los viajes realizados en el libro de servicios, una variable binaria *E* que toma el valor 1 si el viaje

asociado es un expreso y 0 en caso contrario, y una función *dist*, que retorna la distancia asociada a cada viaje.

$$\sum_{t}^{t \in V} E_t.dist(t)$$

La cantidad de vehículos, se define en términos de la solución encontrada, S, y la función *cantV* que retorna la cantidad de vehículos utilizados en la misma.

$$cantV \quad (S)$$

La cantidad de jornales, a su vez, se define en términos del conjunto D, de los jornales incluidos en la solución.

$$\sum_{d}^{d \in D} 1$$

La cantidad de horas totales, se define como sigue, utilizando el mismo conjunto D y la función *duración*.

$$\sum_{d}^{d \in D} duración(d)$$

Finalmente, la cantidad de horas que faltan en los turnos que no alcanzan una duración mínima, se contabiliza a partir de la constante *DMIN* y la variable binaria M_d que toma el valor 1 si el turno *d* no alcanza la duración mínima y 0 en caso contrario.

$$\sum_{d}^{d \in D} M_d.(DMIN - duración(d))$$

Todas las funciones antes mencionadas, se combinan linealmente utilizando los coeficientes $X_{1..5}$ para escalarlas y ponderar su importancia relativa. Los valores de los coeficientes son especificados como datos de entrada de acuerdo a los intereses deseados. La función, entonces, queda como sigue:

$$X_1.\sum_{t}^{t \in V} E_t.dist(t) + X_2.cantV(S) + X_3.\sum_{d}^{d \in D} 1 + X_4.\sum_{d}^{d \in D} duración(d) + ..$$
$$.. + X_5.\sum_{d}^{d \in D} M_d.(DMIN - duración(d))$$

Lo cual, puede ser reducido a la siguiente función:

$$X_1.\sum_{t}^{t\in V} E_t.dist(t) + X_2.cantV(S) + \sum_{d}^{d\in D}(X_3 + X_4.duración(d) + X_5.M_d.(DMIN - duración(d)))$$

Nuestro objetivo principal, será entonces, minimizar el valor de la función mencionada, por lo cuál podríamos tomar a la función de *fitness* como el inverso de la misma.

3.5. Ajuste Dinámico de la Probabilidad de Mutación

Se implementó además en el algoritmo un mecanismo que permite adaptar dinámicamente la probabilidad de mutación. La idea subyacente es la siguiente: Cuando vemos que luego de varias generaciones el *fitness* promedio de la población se mantiene en una franja determinada, se intenta incrementar la probabilidad de mutación, intentando así evadir la caída en óptimos locales, y buscando una exploración más amplia del espacio de soluciones.

El mecanismo opera entonces de la siguiente manera, si el *fitness* promedio de la población se mantiene en una franja correspondiente a cierto porcentaje configurable del *fitness* durante una cantidad configurable de generaciones, entonces se incrementa la probabilidad de mutación en una fracción configurable de lo que le falta para llegar a 1.

Luego, cuando el *fitness* promedio de la población sale de la franja se vuelve a decrementar la probabilidad de mutación en las mismas proporciones.

3.6. Escenario de Configuración y Parámetros Principales

3.6.1. Escenario de Configuración

Como hemos mencionado, es importante para obtener soluciones de buena calidad en tiempos razonables, someter algunos parámetros de configuración a pruebas (*tests*) de calibración. En esta sección, se presentarán las pruebas de configuración realizadas para el problema, de manera que el lector pueda concluir cuáles son los mejores valores a utilizar en cada parámetro para la ejecución del algoritmo.

En ésta sección se fijaron los coeficientes ($X_{1..}X_5$) de la función de *fitness*, para obtener un valor del mismo. Estos se definieron utilizando estimados del valor económico de cada una de las funciones involucradas. Cabe mencionar la importancia de dichos coeficientes dada la heterogeneidad de los factores que influyen en la función de *fitness*, y la necesidad de procurar normalizar sus valores de manera de lograr un comportamiento de la función acorde a los intereses deseados.

Los coeficientes se fijaron en los siguientes valores, los kilómetros expresos se multiplican por 100, asumiendo que el costo asociado a realizar un kilómetro expreso es

aproximadamente $100, del mismo modo, la cantidad de vehículos se multiplica por 2000, la cantidad de turnos por 300, la cantidad de horas trabajadas por 120 y la cantidad de minutos por debajo del jornal mínimo se penaliza con 20. El jornal se define en 8 horas, y el jornal mínimo en 4 horas, de manera que si un jornal se planifica para durar menos de 4 horas, el mismo es penalizado en la función de *fitness*.

A continuación, se mencionan los parámetros sobre los cuales se trabajará en el ajuste.

3.6.2. Método de Selección

El tipo de selección es un parámetro que debe ser ajustado. Se han realizado las correspondientes pruebas utilizando *Rueda de Ruleta* y selección *Estocástica Universal*, operadores que se presentan en las secciones 3.6.2.1 y 3.6.2.2 respectivamente. La selección es un operador del algoritmo genético simple que es muy importante, en tanto que determina cuán fuerte será la presión de selección asociada al método, una fuerte presión de selección, puede ocasionar que la búsqueda converja prematuramente en un óptimo local, una presión de selección muy baja, puede causar que la búsqueda avance muy lentamente.

3.6.2.1. Rueda de Ruleta

El operador de selección por rueda de ruleta[GOLD] opera asignando a cada individuo una probabilidad de selección proporcional al *fitness* relativo del mismo.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Éste es un operador que tiene una alta presión selectiva, sobretodo en los casos en los cuales hay un individuo con *fitness* demasiado alto en comparación con el resto de la población, sin embargo, si el *fitness* de la población no tiene grandes diferencias, el operador permite seleccionar también individuos de bajo *fitness* relativo –aunque con probabilidad baja- y no es un operador elitista, desde el punto de vista que no garantiza la permanencia en la población de los individuos de alto *fitness*.

El mecanismo utilizado por el operador es el siguiente: se colocan en un segmento de recta todos los individuos, y se realiza un sorteo en el cual todos tienen probabilidad proporcional a su *fitness*. La Figura 3.6.2.1 muestra un ejemplo de la selección de 6 individuos, en una población de 10, a través del sorteo (*trial*) de 6 muestras de una variable aleatoria con distribución uniforme en el intervalo [0, 1].

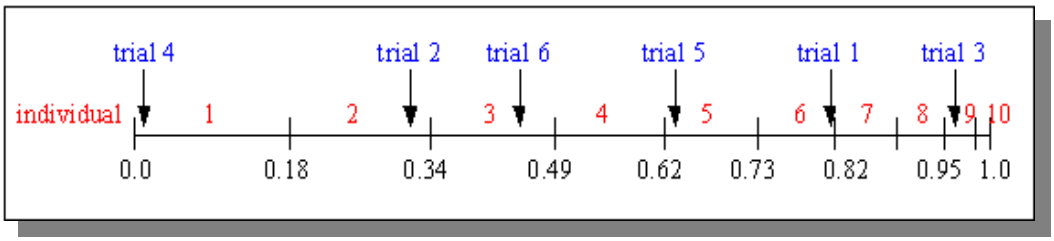

Fig. 3.6.2.1 Ejemplo de selección por rueda de ruleta.

3.6.2.2. Selección Estocástica Universal

La selección estocástica[GOLD] es similar al método de rueda de ruleta con la excepción de que no se realizan varios sorteos independientes, sino que se realiza un sorteo solamente y se definen punteros (*pointers*) equidistantes para seleccionar el resto de los individuos. La Figura 3.6.2.2 muestra un ejemplo de selección para 6 individuos donde la distancia entre los punteros es de 1/6.

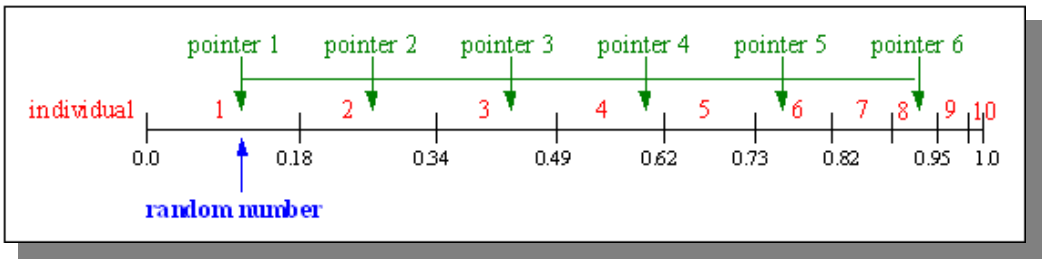

Fig. 3.6.2.2. Ejemplo de selección estocástica universal

3.6.3. Probabilidad de Cruzamiento

El operador de cruzamiento, busca garantizar la explotación de buenas secciones del espacio de búsqueda. Ajustaremos la probabilidad de cruzamiento utilizando dos valores que evaluamos como apropiados: 0,5 y 0,7.

3.6.4. Probabilidad de Mutación

El operador de mutación, busca introducir diversidad y explorar nuevas secciones del espacio de búsqueda (soluciones), realizaremos pruebas utilizando dos valores considerados adecuados, los mismos son 0,1 y 0,2.

3.6.5. Tamaño de la Población

El tamaño de la población es un parámetro que también buscaremos ajustar, los tamaños que contrastaremos serán 60 y 100.

3.6.6. Porcentaje de Mutación

Tal y como fue explicado en la sección 3.3.3, la mutación tiene un parámetro adicional, el cuál habla del porcentaje de la solución que será afectado en la mutación. Éste parámetro también debe ser ajustado, realizaremos las pruebas utilizando los valores 0,1 y 0,05.

3.7. Pruebas

Se presentan en la Figura 3.7.1 la tabla de resultados de las pruebas de ejecución con los valores propuestos. Estos son el resultado de ajustar los parámetros especificados en cada caso. Se realizan 30 ejecuciones independientes con cada juego de parámetros y se presenta el valor del mejor costo y tiempo de ejecución obtenido en ellas.

La duración que se presenta en la columna *tiempo* está expresada en segundos, y corresponde al promedio de 10 ejecuciones independientes (cada ejecución toma aproximadamente un décimo del valor que figura en la tabla). Todas las pruebas se realizan utilizando el caso de *Hector* (66 viajes, 3 ciudades), especificado en la sección 7.2 para el escenario definido en la sección 7.1.2, utilizando como reglas laborales 8 horas de duración para cada jornal con un descanso de 30 minutos o tres descansos de 15 minutos.

Caso	#(población)	selección	p(cruzamiento)	p(mutación)	%(mutación)	costo	tiempo
1	60	rr	0,5	0,1	0,1	50700	92
2	60	rr	0,5	0,1	0,05	46640	57
3	60	rr	0,5	0,2	0,1	46640	93
4	60	rr	0,5	0,2	0,05	56540	59
5	60	rr	0,7	0,1	0,1	50700	99
6	60	rr	0,7	0,1	0,05	46640	64
7	60	rr	0,7	0,2	0,1	56540	99
8	60	rr	0,7	0,2	0,05	56540	63
9	60	e	0,5	0,1	0,1	56570	100
10	60	e	0,5	0,1	0,05	46640	64
11	60	e	0,5	0,2	0,1	51050	99
12	60	e	0,5	0,2	0,05	56570	64
13	60	e	0,7	0,1	0,1	50700	105
14	60	e	0,7	0,1	0,05	46640	68
15	60	e	0,7	0,2	0,1	50700	104
16	60	e	0,7	0,2	0,05	46640	70
17	100	rr	0,5	0,1	0,1	46860	93
18	100	rr	0,5	0,1	0,05	50700	58
19	100	rr	0,5	0,2	0,1	46640	93
20	100	rr	0,5	0,2	0,05	46860	59
21	100	rr	0,7	0,1	0,1	46640	96
22	100	rr	0,7	0,1	0,05	46640	62
23	100	rr	0,7	0,2	0,1	46640	95
24	100	rr	0,7	0,2	0,05	46640	61
25	100	e	0,5	0,1	0,1	46640	100
26	100	e	0,5	0,1	0,05	46640	65
27	100	e	0,5	0,2	0,1	46860	103
28	100	e	0,5	0,2	0,05	50700	64
29	100	e	0,7	0,1	0,1	46640	106
30	100	e	0,7	0,1	0,05	46640	69
31	100	e	0,7	0,2	0,1	46640	106
32	100	e	0,7	0,2	0,05	46860	69

Fig. 3.7.1. Tabla de calibración del caso *Hector* (*rr-Rueda de Ruleta, *e-Selección Estocástica).

Si bien el problema no tiene instancias estándar contra que compararlo, de la tabla podemos deducir, que hay un valor de costo, que resulta asintótico para el sistema, el cuál nunca obtiene valores por debajo del mismo (46640). Asimismo, podemos afirmar que el incremento del tamaño de la población a 100 individuos hace que, casi siempre, se obtengan mejores costos con mínimas diferencias en el tiempo de ejecución. Puntualmente, la diferencia es de seis centésimas entre el promedio de tiempo para 60 y 100 individuos.

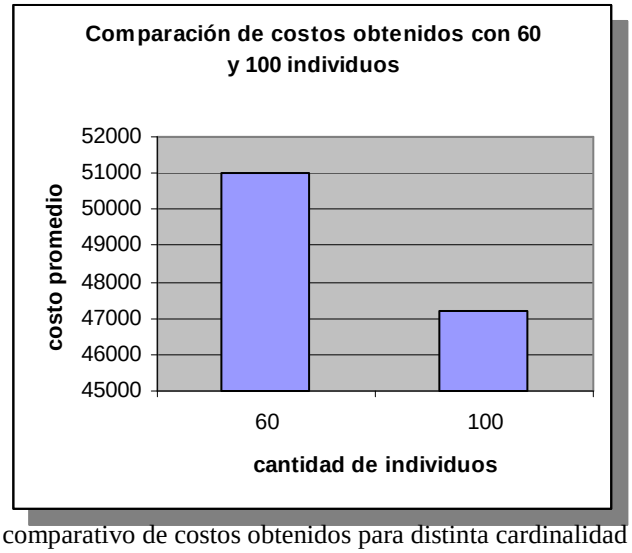

Fig. 3.7.2. Gráfico comparativo de costos obtenidos para distinta cardinalidad de población.

Otro resultado que se desprende de la tabla, es que tener un 10% de porcentaje de mutación, incrementa cerca de un 50% el tiempo de ejecución, en relación con mantenerlo en 5%. Sin embargo, los resultados en términos de costo no varían demasiado. En la Figura 3.7.3. se presenta el tiempo de ejecución promedio variando el porcentaje de mutación, en estos casos, se nota que los valores promedio son significativamente distintos, mientras que el costo promedio varía solamente en 164 unidades, lo cual no es significativo, ya que hablamos de costos del entorno de 48000 unidades.

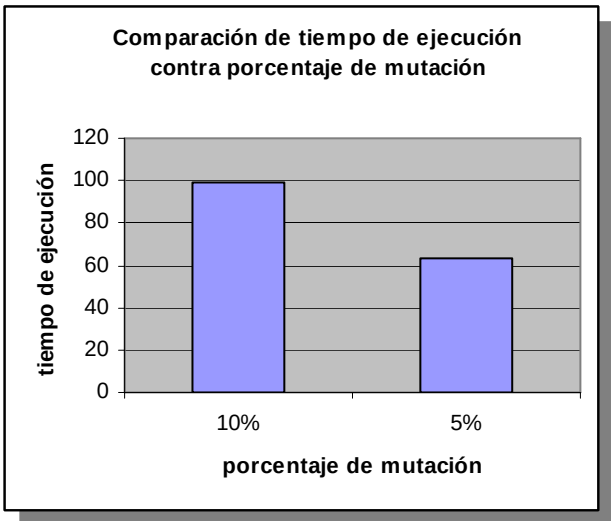

Fig. 3.7.3. Gráfico comparativo del tiempo de ejecución en función del porcentaje de mutación del individuo.

En cuanto a los mecanismos de selección, no se observan grandes diferencias. Sin embargo, se puede afirmar que hay una pequeña mejora en los costos que se obtienen con selección estocástica. Esta pequeña mejora, tiene como contrapartida el hecho de que se empeora sensiblemente el tiempo de ejecución. Aún así, y entendiendo el impacto sobre la performance, se utilizará el método de selección estocástico.

A continuación se presentan las gráficas que estudian los resultados de los distintos métodos de selección.

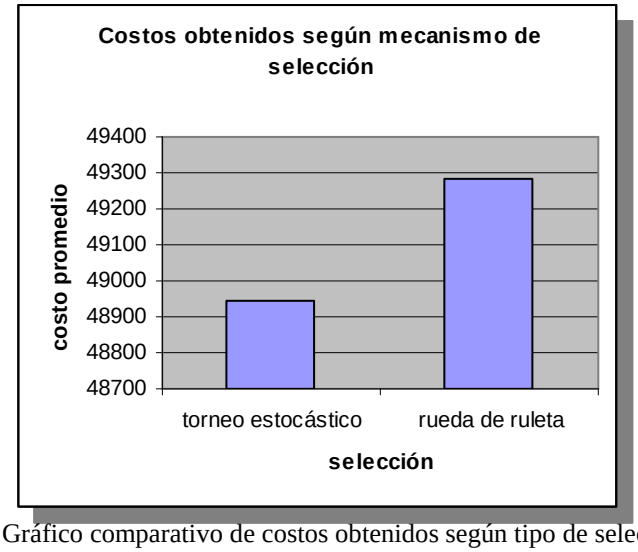

Fig. 3.7.4. Gráfico comparativo de costos obtenidos según tipo de selección.

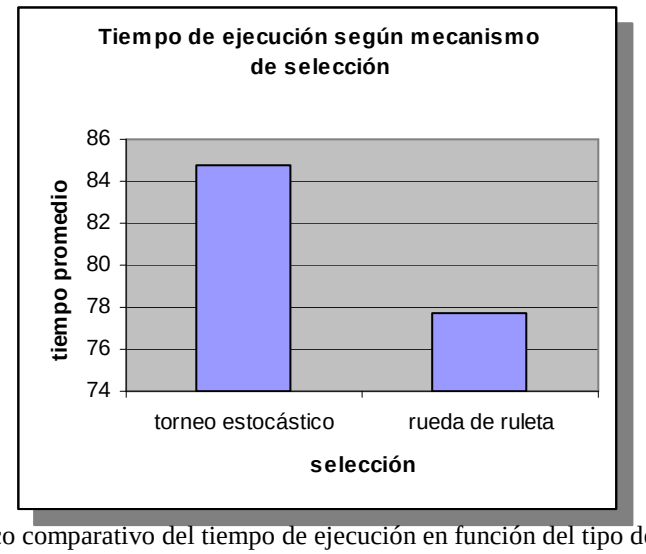

Fig. 3.7.5. Gráfico comparativo del tiempo de ejecución en función del tipo de selección.

Algo similar ocurre con la probabilidad de cruzamiento, definirla en 0,7 mejora levemente los resultados en términos de costos, pero también empeora levemente la performance del sistema en términos del tiempo de ejecución. Se muestran a continuación las gráficas que presentan éstos resultados.

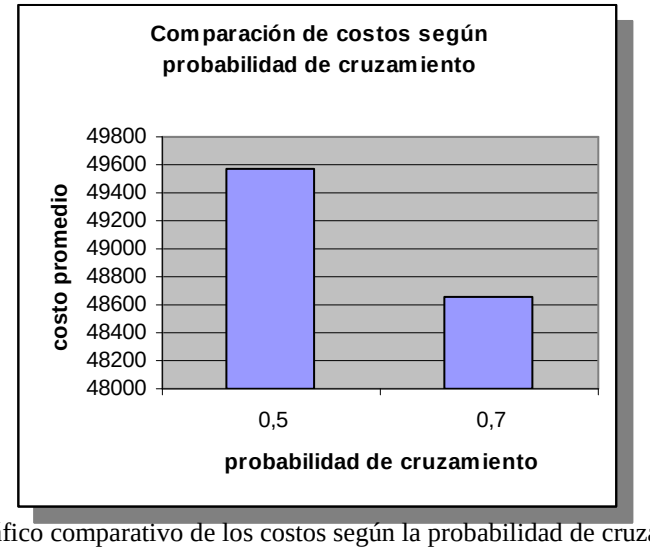

Fig. 3.7.6. Gráfico comparativo de los costos según la probabilidad de cruzamiento.

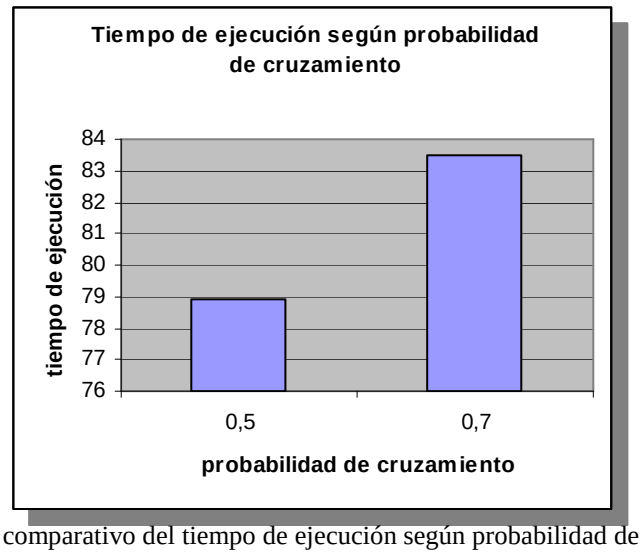

Fig. 3.7.7. Gráfico comparativo del tiempo de ejecución según probabilidad de cruzamiento

En el caso del porcentaje de mutación, no ocurre lo mismo, se mejora levemente el costo promedio utilizando la probabilidad de mutación en 0,1; y la diferencia en cuanto a términos de tiempo de ejecución es de solamente dos décimas de segundo.

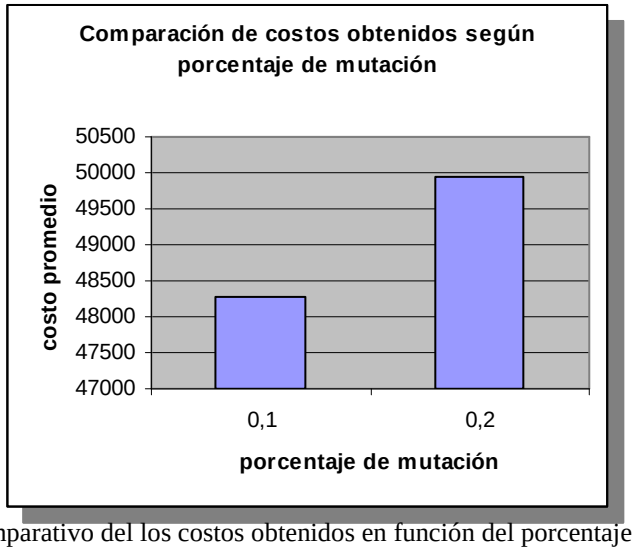

Fig. 3.7.8. Gráfico comparativo del los costos obtenidos en función del porcentaje de mutación del individuo

Finalmente, y luego de dadas todas las razones, los valores que se sugieren utilizar, serían los presentados en la Figura 3.7.9.

Tamaño de la población = 100 individuos

Tipo de selección = Selección Estocástica

Probabilidad de cruzamiento = 0,7

Probabilidad de mutación = 0,1

Porcentaje de mutación = 0,05

Fig. 3.7.9. Parámetros sugeridos a partir de la calibración.

Todas las pruebas fueron realizadas en una máquina virtual VMware Server Console[VMWARE] 1.0.5 con 2 procesadores y 512 Mb de memoria asignados corriendo sobre un equipo Dell Latitude E6400[DELL] con procesadores Intel[INTEL] Core 2 Duo de 2,26 GHz y 4Gb de memoria RAM.

3.8. Pruebas Adicionales

Otros operadores también requieren configuración, veremos a continuación cuál es el efecto que se percibe al modificar dichos operadores. Todos los resultados que se presentan se generaron utilizando la mejor configuración explicada en la sección 3.7, y son el promedio de 30 pruebas independientes.

3.8.1. Tipo de inicialización

Como se detalló en la sección 3.2, se han desarrollado tres modos de generación de soluciones iniciales: generación aleatoria, generación secuencial, y generación utilizando el mecanismo de entronques. El sistema desarrollado permite que la población inicial se genere utilizando un porcentaje de cada uno de los métodos, a continuación se presentaran resultados generados utilizando diversas configuraciones de inicialización.

caso	p(aleatoria)	p(secuencial)	p(entronques)	costo	tiempo
1	1	0	0	120510	66
2	0	1	0	No inicia	No inicia
3	0	0	1	No inicia	No inicia
4	0,333	0,333	0,333	50700	63
5	0,5	0,25	0,25	51050	65
6	0,25	0,5	0,25	46640	64
7	0,25	0,25	0,5	46640	63

Como vemos en la tabla anterior, inicializando únicamente con el modo aleatorio, las soluciones que se obtienen son sensiblemente peores que las que se obtienen utilizando algunas de las otras técnicas que hacen uso de información del problema. En el caso secuencial, puntualmente, este conocimiento adicional surge del análisis del problema, y en el caso de los entronques, ese conocimiento fue provisto por parte de los integrantes del área de planificación de la empresa COPSA, y constituye parte de su experiencia planificando los viajes.

Por otra parte, también cabe mencionar que cuando se selecciona solamente un modo de inicialización no aleatorio, el sistema puede no obtener soluciones iniciales. Esto ocurre porque los otros modos de inicialización encadenan los viajes de tal manera que, de acuerdo a las restricciones definidas el módulo de reglas laborales no encuentra espacios para introducir los descansos de los trabajadores tal como se indica mediante los parámetros dados en la sección 3.7.

Finalmente, podemos notar, que combinando la inicialización aleatoria con los otros modos se obtienen mejores poblaciones desde el punto de vista de diversidad en tiempos prácticamente iguales.

Ya que la inicialización por el método de entronques contiene conocimiento basado en las técnicas utilizadas y probadas durante años por una empresa operando en el mercado, la configuración sugerida para la inicialización es la que se plantea a continuación:

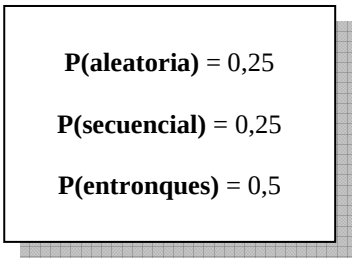

Fig. 3.8.1. Configuración sugerida para métodos de inicialización.

3.8.2. Tipo de Cruzamiento

También se ha detallado en la sección 3.3.2 que se han implementado dos operadores diferentes de cruzamiento, y se generó un parámetro que se utiliza para decidir qué porcentaje de veces se utiliza uno u otro operador.

Se presentan, en la tabla que vemos a continuación, el resultado de utilizar los dos modos de cruzamiento.

caso	C1	C2	costo	tiempo
1	1	0	46640	63
2	0	1	46640	68
3	0,5	0,5	46640	65
4	0,7	0,3	46640	66
5	0,3	0,7	46860	67

La columna C1, representa el cruzamiento que se realiza en un solo punto (ver sección 3.3.2), mientras que la columna C2 representa al operador de cruzamiento a partir de un punto. De la tabla se desprende que el uso de uno u otro operador no afecta la calidad de las soluciones obtenidas, mientras que el operador de cruzamiento a partir de un punto incrementa un poco el tiempo de ejecución, lo cual es razonable, a raíz de que el cruzamiento a partir de un punto tiene un costo computacional más alto. Esto se debe a que modifica los viajes a partir de un punto, en vez de modificar solamente el viaje de un punto seleccionado.

La configuración elegida para el caso es la configuración en la cual solamente se utiliza el operador de cruzamiento solamente en un punto.

3.8.3. Ajuste Dinámico de la Probabilidad de Mutación

En la sección 3.5 se comentó acerca de la posibilidad de utilizar un *feature* que se desarrolló adicionalmente al algoritmo genético puro. El coeficiente adaptativo para la probabilidad de mutación puede ser habilitado y deshabilitado; además de esto, puede ser modificado el umbral que determina cuándo una evolución está estancada, y cuánto se modifica la probabilidad de mutación en caso de que la evolución se mantenga dentro de ese rango.

A continuación, mostraremos resultados que se logran habilitando el *feature* de probabilidad de mutación adaptativa y configurándolo de distintas maneras.

Lo que se define en la tabla como umbral, es un valor porcentual que indica que si el *fitness* se mantiene durante cierta cantidad de generaciones en la franja que determina el *fitness* promedio actual más menos un cierto porcentaje del mismo, entonces la probabilidad de mutación se incrementa, sino se retrae. Esto responde a la idea de que si el *fitness* se encuentra estancado durante varias generaciones en una franja pequeña, entonces, intentamos incrementar un poco la evolución, con el fin de evitar la caída en óptimos locales.

caso	generaciones	umbral	costo	tiempo
deshabilitado	No aplica	No aplica	46640	63
habilitado	5	5%	46640	67
habilitado	10	5%	46640	68
habilitado	5	10%	46640	69
habilitado	10	10%	46640	69
habilitado	5	20%	58410	67
habilitado	10	20%	56570	64

De la tabla se desprenden algunas observaciones, la primera de ellas es, que si el *feature* se habilita, el tiempo de ejecución se incrementa. Por otro lado, vemos que si se define un umbral demasiado grande, la probabilidad de mutación puede crecer más de lo deseado, y, por tanto, generar soluciones de baja calidad en términos del *fitness*.

Como conclusión del capítulo, presentamos a continuación un listado de los parámetros que se han calibrado, y los valores sugeridos para cada uno de ellos.

Tamaño de la población = 100 individuos

Tipo de selección = Torneo estocástico

Probabilidad de cruzamiento = 0,7

Probabilidad de mutación = 0,1

Porcentaje de mutación = 0,05

P(inicialización aleatoria) = 0,25

P(inicialización secuencial) = 0,25

P(inicialización entronques) = 0,5

cruzamiento = en un solo punto

probabilidad de mutación adaptativa = deshabilitada ó habilitada, definiendo rangos pequeños.

Fig. 3.8.3. Valores sugeridos para la ejecución.

4. Integración de Reglas Laborales

4.1. Antecedentes

Siendo los vehículos y la tripulación los recursos principales en problemas de planificación operacional en empresas de transporte público, particularmente de ordenamiento y asignación de tripulación, las reglas laborales que restringen la asignación de tripulación tienen gran influencia en los costos y resolución de los problemas.

Existen múltiples experiencias en cuanto al modelado y resolución de problemas relativos a la asignación de tripulación. Por ejemplo DSP (*Duty Scheduling Problem*) y BDSP (*Bus Driver Scheduling Problem*)[PAIX] los cuales utilizan distintas técnicas donde se incluyen algoritmos genéticos[GABDSP]. Este tipo de problemas, típicamente involucran normas laborales complejas y particulares a cada realidad, donde los costos que se buscan minimizar están sujetos a la aplicación de leyes, tarifas, políticas y otras posibles restricciones[HAO08]. Para ello, las formulaciones usualmente buscan abstraer esta problemática de modo de poder plantear dos objetivos básicos en búsqueda de una asignación de tripulación óptima restringida al conjunto de normas. Por un lado, la minimización de la cantidad de *duties*, lo cual implicará menos jornales de trabajo y en segundo lugar la minimización de las horas empleadas para cumplir con los viajes especificados. Dada la complejidad que implica este tipo de problema por sí solo, usualmente modelado como *Set Partitioning Problem*[SETPA] (*NP-HARD*), y sumado al problema de ordenamiento de vehículos, el enfoque para la resolución de ambos como conjunto se planteaba de forma secuencial, manejando cada uno de ellos como subproblemas consecutivos. Fue en los años ochenta cuando se comenzó a contemplar la resolución del problema integrado fomentado por críticas al enfoque secuencial como la planteada por Ball et al (1983)[BALL]. De todos modos, algunas de las propuestas integradas realmente eficientes (no necesariamente algoritmos genéticos) no aparecieron hasta 1995, como la planteada por Freling et al[IAVCS] y Gaffi, Nonato (1999)[IAECVSP]. Entre las formulaciones más recientes (2008) y ajustada al enfoque de la resolución propuesta se encuentra la realizada por Jin-Kao Hao, Benoit Laurent (2008)[HAO08] donde su resolución se basa en técnicas como GRASP (*Greedy Randomized Adaptive Search Procedure)* e ILS (*Improvement by Local Search).*

4.2. Modelado de Reglas

Es claro que dada la naturaleza de las normativas pertinentes a esta realidad, las mismas no están sujetas únicamente a normas laborales existentes, sino que también al constante cambio y posibilidad de nuevas propuestas y variantes que satisfagan mejor aún los intereses de las partes involucradas.

Es así que la necesidad de actualización permanente y flexibilidad en la especificación de tales conceptos, se convierte en uno de los requerimientos más relevantes en la integración de estas restricciones a la búsqueda de soluciones. La necesidad y opción de poder modelar nuevas restricciones que permitan buscar o mejorar soluciones durante la ejecución del algoritmo debería de ser atractiva para

cualquier compañía que cuente con ciertos márgenes de flexibilidad en la asignación de su flota y tripulación, donde experimentar con distintas combinaciones de valores o restricciones modelando distintos escenarios podría llevar a explorar nuevas asignaciones en búsqueda de mejores resultados.

Para el modelado de reglas laborales se propone la utilización de un lenguaje definido en la sección 4.2.1 que permite la especificación de tales conceptos.

4.2.1. Lenguaje

Se plantea la utilización de un lenguaje que permite una primera aproximación de los conceptos manejados en la realidad, a un nivel de especificación lógica y formal que pueda ser interpretado para la validación de las soluciones buscadas sin ambigüedad.

El lenguaje comprende componentes lógicos de primer orden donde se incluyen como base: operadores unarios, binarios, parámetros, identificadores y funciones. De este modo, se provee de un lenguaje simple, estándar y extensible (como se verá en la sección: 4.2.5), para el cual basta con conocer la sintaxis definida de modo de poder agregar o modificar dinámicamente nuevas reglas. Los detalles sobre la especificación de reglas se muestran en el anexo C. Pero a modo de comprensión la misma se basa en la gramática detallada en la Figura 4.2.1.

```
Gramática del Lenguaje

WorkerRule  ::=   Rule

Rule        ::=   Op_Binario(Rule,Rule)   |
                  Op_Unario(Rule)         |
                  Op_Comp(Param,Param)    |
                  FUNCION()

Op_Comp     ::=   GREATER |
                  LESSER  |
                  EQUAL

Param       ::=   IDENTIFICADOR |
                  CONSTANTE

Op_Binario  ::=   OR |
                  AND

Op_Unario   ::=   NOT
```

Fig. 4.2.1. Gramática del Lenguaje – Módulo Reglas Laborales (WRM)

```
<WorkerRule>
OR(GREATER(SHIFTDURATION,"240"),
       LESSER(MINSHIFTKMRELATION,"0.6"))
</WorkerRule>
```

Fig. 4.2.2. Ejemplo de definición de regla laboral.

La Figura 4.2.2. muestra un ejemplo de definición de una regla laboral compuesta por el operador binario *OR*, operadores de comparación (*GREATER* y *LESSER*) e identificadores y constantes. La misma podría leerse como: los turnos deberán tener duración (*SHIFTDURATION*) superior a 240 minutos o la relación de kilometraje *kmExpresos/kmTotal* (*MINSHIFTKMRELATION*) deberá ser menor a 0.6.

4.2.2. Identificadores y Constantes

Estos elementos representan terminales de la gramática con la diferencia de que mientras las constantes representan valores concretos (simples o vectoriales), los identificadores modelan conceptos definidos y particulares de la realidad planteada. Los mismos podrán representar entidades o conceptos particulares, propios de una solución o relativos a cualquier elemento o conjunto de elementos de los comprendidos en la misma.

4.2.3. Funciones

El uso de funciones pretende facilitar la combinación de elementos de manera de poder agregar validaciones que requieran lógica y/o poder de cálculo así como funcionalidades más complejas; donde disponer de las bibliotecas y propiedades de un lenguaje de programación sea de gran ayuda o un requisito.

4.2.4. Operadores

Los operadores brindan la posibilidad de establecer relaciones lógicas entre cualquiera de los componentes. En particular se definen los operadores clásicos de negación (*NOT*), conjunción (*AND*) y disyunción (*OR*) que incluyen un conjunto de conectivos funcionalmente completo abriendo así el espectro de especificación de restricciones, permitiendo la definición de cualquier función de verdad entre los componentes del lenguaje.

4.2.5. Extensibilidad

La extensibilidad en el modelado de reglas no viene dada únicamente por la posibilidad de definir, combinar o agregar reglas a partir de componentes ya existentes, sino que también se anticipa la posibilidad de extensión y creación de nuevos componentes del modelo que permitan proveerle al usuario un modelo tan nutrido como se desee.

Considerando como requerimientos principales, por un lado la necesidad implícita de poder extender un lenguaje en lo que concierne al relacionamiento entre componentes (nuevos operadores, relaciones o condiciones), y por otro lado, la necesidad de poder referirse a conceptos, valores o propiedades de las entidades presentes en la problemática planteada, es que se definen interfaces para cada uno de los componentes definidos, y mediante la implementación de las mismas, se podrán integrar nuevos componentes. Los detalles sobre la incorporación de estos se especifican en el Anexo C, y se incluye una guía en el Anexo G, pero básicamente el mecanismo consta de la edición de archivos en formato XML[XML] que permiten dinámicamente la incorporación y categorización de nuevas bibliotecas dinámicas[DLL](*.so*), que incorporen tantos componentes como se desee para su eventual uso en la definición de reglas.

```
<Library type=cond>
      <path>wrm/Logical.so</path>
      <objects>
            <obj>AND</obj>
            <obj>OR</obj>
            <obj>NOT</obj>
      </objects>
</Library>
```

Fig. 4.2.5. Ejemplo de incorporación de la biblioteca Lógica (*Logical.so*), con 3 componentes (*AND, OR, NOT)*

Al integrar una nueva biblioteca, sus componentes definidos (a través de elementos *obj* en el archivo de configuración) serán incorporados a la gramática del lenguaje de acuerdo a la interfaz que cada uno de ellos implemente. Al ser cargada la biblioteca, cada uno de ellos será clasificado según su tipo e incorporado como operadores, terminales, identificadores, etc., según corresponda.

4.3. Interacción con el Algoritmo

Dado que el enfoque de la solución propuesta busca la integración de las reglas laborales de modo de optimizar el ordenamiento de la flota junto con la asignación de tripulación, se plantea una interacción constante entre el algoritmo genético y la validación de reglas, de modo de estudiar la factibilidad lo más prematuramente posible. Para ello, se incluye el estudio de factibilidad pertinente a las reglas laborales y a los

métodos de generación de soluciones iniciales en cada una de las asignaciones realizadas, así como también en los operadores de cruzamiento y mutación. Es decir, que se buscará validar cada una de las decisiones o cambios realizados sobre cada individuo durante la generación y evolución del mismo y no esperar a realizarlo sobre la población completa. Esto permite mayor integración de las reglas laborales al proceso evolutivo descartando generaciones de individuos no factibles.

```
Solucion run(){
      InicializarConWRM(P(0));
      generacion=0;
      mientras (noCriterioParada) hacer
            Evaluar (P(generacion));
            Padres = Seleccionar(P(generacion))
            Hijos = AplicarCruzamientoConWRM(PadreS)
            Hijos = AplicarMutacionConWRM(Hijos)
            NuevaPob = Reemplazar(Hijos, P(generacion))
            generacion++;
            P(generacion) = nuevaPob;
      fin;
      return mejorSolucion(P);
}
```

Fig. 4.3. Pseudocódigo de ejecución del algoritmo integrando las reglas laborales

4.3.1. Conceptos Claves

Durante el estudio del problema y relevamiento de antecedentes se aprecia la existencia de conceptos propios a la realidad, cuya relevancia suele ser fundamental e independiente a una instancia particular, o a la variación que pueda existir en valores o unidades consideradas. Contemplar dichos conceptos en alguna o varias etapas, permiten al algoritmo hacer uso del conocimiento sobre la realidad manejada. De este modo, y teniendo en cuenta la parametrización de valores correspondiente, la posibilidad de considerar las propiedades de dichos conceptos, permite modelar y aplicar el conocimiento a las técnicas y algoritmos utilizados con el objetivo de obtener mejores resultados.

Dicho lo anterior, se propone embeber conceptos comunes a diferentes realidades, escenarios e instancias de la problemática en cuestión, de modo de poder sacarle provecho a la información de los mismos durante la etapa de ordenamiento y asignación con el objetivo de buscar soluciones teniendo en cuenta dicha información y no simplemente validar contra la misma. Esto no solo mejora la eficiencia y los costos de ejecución en muchas de las técnicas empleadas, como ser las de operadores genéticos, ya que permite la obtención de soluciones factibles de forma más sencilla, sino que también permite mejorar velozmente los resultados de la función de *fitness* de la población, lo cual contribuye a mejorar las soluciones. A continuación, en las

secciones 4.3.1.1 y 4.3.1.2, se describen los conceptos que se identifican y asumen conocidos con los motivos y objetivos previamente descritos.

4.3.1.1. Jornales y Descansos

Estos conceptos representan quizás las restricciones más básicas necesarias, comunes y presentes en las realidades referentes al problema de asignación de tripulación, donde existan normas laborales independientemente del país o región. La limitante implícita ligada al uso de recursos humanos en cuanto a la limitación de horas disponibles, así como las necesidad de intervalos de descanso, no solo regidos por leyes y normas legales sino también fisiológicas, implican factores determinantes a la hora del armado de servicios. Es por esto, que se decide considerar dichos conceptos ya que formarán parte de la realidad común de este problema y semejantes.

En cuanto a los jornales o turnos, utilizaremos dos conceptos claves. Uno de ellos, es la cantidad de horas deseadas por la empresa como duración de un jornal, el cuál se define como *duración óptima*. El objetivo será aproximar la duración de los jornales a dicho valor, ya que usualmente, el mismo se encuentra definido por reglas laborales y apartarse de éste, implicaría incrementos considerables en los costos de la tripulación, como por ejemplo el pago de horas extras a mayor valor. El otro concepto que se utiliza, es el de un valor que represente la cantidad mínima de horas que justifiquen, en materia de costos, la creación de un nuevo jornal. Esto se debe a que la convocatoria de un nuevo tripulante implica, por normas laborales, el pago de costos fijos por jornal. Cabe mencionar que ambos conceptos no pretenden restringir la factibilidad de las soluciones, sino que se utilizan en busca de aproximar las soluciones a los valores óptimos deseados. La penalización o cuantificación de la importancia de la diferencia que exista entre la duración obtenida y la deseada, podrá ser ponderada en la función de *fitness*, o también considerada en las estrategias durante la generación de turnos.

Las reglas de descanso (*break rules)* modelan en sí, las condiciones para la inclusión de los eventos de *Descanso* en los turnos. Los eventos *Descanso* representan un lapso de tiempo definido con hora de inicio y fin durante el cual, la tripulación podrá descansar y atender necesidades fisiológicas. Usualmente sus duraciones y cantidad están regidas tanto por leyes, acuerdos y políticas que la empresa deberá acatar o considerar. Mediante la definición de estas reglas, se especifican las restricciones necesarias de un turno en cuanto a descansos de tripulación refiere, de modo de cumplir con las normas vigentes. Para cada una de estas reglas, a las cuales denominaremos *breakRules*, se considera: la cantidad de descansos de ese tipo a incluir en un turno (*quantity*), la duración de los mismos (*duration*), el margen de tiempo en que tienen que ser efectivos (*start..end*) y el margen de tiempo que debe existir entre ellos (*gap*), donde la unidad para los tiempos se establece en minutos. Dado un conjunto de *breakRules* definido, se buscará que cada turno cumpla con al menos una de las posibilidades de asignación de descansos.

```
<BreakRule>
      <duration>15</duration>
      <quantity>3</quantity>
      <start>60</start>
      <end>420</end>
      <gap>30</gap>
</BreakRule>
```

Fig. 4.3.1.1. Definición de una regla de descanso. (formato XML)

4.3.1.2. Puntos de Relevo

Los puntos de relevo representan una importante restricción en la factibilidad de las soluciones desde el punto de vista de la generación de turnos de un servicio. Validar la incompatibilidad del relevo entre dos turnos representa una poda muy útil a lo hora de generar los libros de servicios. Dicha información es considerada como conocida y proporcionada junto a los datos de entrada la cual indicará la posibilidad de poder realizar relevos de tripulación en un punto dado, pudiendo determinar la finalización e iniciar nuevos turnos para un servicio. Estos datos son utilizados durante la generación de turnos a modo de validación, donde se verifica que efectivamente los puntos de origen y destino de un turno verifiquen la restricción.

4.3.2. Generación de Turnos

Una vez presentada la gestión de reglas laborales y conceptos claves, se incluye como segundo componente principal relativo a las normativas laborales, la funcionalidad de generación de turnos. Teniendo como objetivo independizar al algoritmo, procurando no acoplar las técnicas de resolución de ordenamiento con las referentes a las laborales, así como encapsular el manejo de lo relativo a dichas reglas, es que ésta funcionalidad es llevada a cabo por el propio módulo de reglas laborales (WRM) y provista al módulo de resolución (GAM) a través de una interfaz.

La técnica definida como estándar (*default*) para la generación de turnos, parte de la asignación de vehículos detallada en el cromosoma de la solución, y considera por procesadas las restricciones referentes al ordenamiento realizadas en el GAM. Es así que se busca realizar para cada vehículo, la generación de turnos procurando minimizar la cantidad de estos, aproximar su duración al valor *óptimo deseado* y cumplir con las reglas de descanso definidas, priorizando la aplicación de una u otra según lo indique la definición de las mismas, como se muestra en el pseudocódigo de la Figura 4.3.2.2.

Es decir, sean D el conjunto de turnos (*duties*) pertenecientes a la solución *S* y *B* el conjunto de reglas de descanso (*breakRules*) se desea:

$$\min\left\{ \#D + \sum_{d \in D} (|\, duracionOptima - duracion(d)\,|) \right\} \;/\; \forall\, d \in D, \exists\, b_i \in B \,/\, b_i.valida(d)$$

Donde *valida()* representa un predicado que indica la posibilidad de aplicación de la regla de descanso *b al turno d*.

El proceso de generación de los turnos, utiliza la asignación de vehículos de modo de validar el servicio de cada uno de estos. Se agrupan los viajes para formar los turnos, procurando aproximarlos a la *duración óptima* fijada (dato de entrada). Durante esta etapa, cada turno generado es procesado para validar el cumplimiento de reglas de descanso y puntos de relevo. Tratándose de dos restricciones de fuerte impacto sobre la factibilidad, su temprana evaluación ahorra y reduce los tiempos y costos de ejecución de procesar soluciones, que a priori, se pueden descartar o evaluar como no factibles. El pseudocódigo de la Figura 4.3.2.1 ilustra los pasos principales para la generación de servicios de cada vehículo.

```
generarTurnos(){
    for each v in Vehicles do
    solFactible = true;
      while (viajesSinAsignarATurno(v) > 0 && solFactible)
          Turno t = armarTurno(v);
          if(!validaBreakRules(t)||!validaReliefPoints(t))
               solFactible = isAsignacionFactible(v);
          else
              agregarTurnoALibroSvc(v,t)
        end;
    end;
}
```

Fig. 4.3.2.1 Pseudocódigo de generación de turnos.

Procurando aplicar y hacer uso de la información disponible, como ser el conocimiento de conceptos ya definidos, el proceso de generación de turnos integra las reglas de descanso (*breakRules*) para la generación de los eventos *Descanso* de cada turno. Es decir, que para la construcción del mismo, una vez definidos los viajes correspondientes a éste, y utilizando los datos conocidos sobre *duraciones óptimas*, se busca la validación de alguna de las reglas definidas, generando los descansos correspondientes, lo que implica una validación implícita de alguna de estas en dicho proceso.

El proceso de validación estándar (*default*) de puntos de relevo es más simple, ya que consta de la evaluación de predicados sobre los lugares de inicio y fin de cada turno que validarán o no el servicio. Particularmente, para la realidad considerada, se contempla además la alternativa de especificar los denominados “relevos al vuelo”, que implican la posibilidad de realizar los relevos de tripulación para un vehículo en un determinado momento de un servicio, sin necesidad que este se encuentre en un depósito (o terminal). Esto brinda mayor granularidad a la hora del armado de turnos, ya que se puede ajustar la asignación de viajes cercanos o que excedan los “limites” de duración más fácilmente, por ejemplo asignando nueva tripulación. Pese a que la técnica por defecto considera dichos puntos solamente a modo de validación, se anticipa (ofreciendo mecanismos de extensión) la posibilidad del uso de técnicas de mejora basadas en dicha información. Una de ellas, podría ser, que ante la finalización de un turno (intermedio de un servicio) en un punto que no fuera de relevo, busque alguno cercano con algún algoritmo del tipo BFS (*Breadth First Search*), y se analice la posibilidad de generación de un viaje expreso hacia el mismo, que permita la finalización del turno como corresponde (en un punto de relevo).

```
Boolean validaBreakRule(Turno t){
  breakRule=first(BreakRules);
  valida=false;
  while (existe breakRule && !valida)
        valida = validaBreakRule(turno, breakRule);
        breakRule= next(breakRule);
  end
  return valida;
}
```

Fig. 4.3.2.2 Pseudocódigo de validación de turnos.

```
Boolean validaReliefPoint(Turno t){
  inicio = getPrimerViajeTurno(t).getOrigen();
  destino = getUltimoViajeTurno(t).getDestino();
return isReliefPoint(destino)&&isReliefPoint(destino);
```

Fig. 4.3.2.3 Pseudocódigo de validación de puntos de relevo.

5. Incorporación de Extensiones

Teniendo presente la importancia del requerimiento de extensibilidad del sistema a construir, así como la necesidad que la problemática plantea en cuanto a la continua adaptación y mejora que puedan requerir distintos escenarios e instancias del problema, se provee un entorno para la fácil incorporación de extensiones y/o mejoras en los puntos clave del sistema, el mismo se denomina Módulo de Extensiones (EM).

Este módulo centraliza la gestión de posibles extensiones y procesos de mejora para un conjunto de puntos clave identificados como decisivos, tanto en el módulo del algoritmo genético (GAM) así como en el módulo de reglas laborales (WRM). Análogamente al mecanismo de extensiones del lenguaje, mencionadas en la sección 4.2.5, se provee una plataforma que permite de manera sencilla, mediante la especificación en un archivo de configuración, la incorporación de nuevos componentes según el interés del usuario.

Entre los puntos clave que se identifican como procesos o funciones fundamentales del sistema, se incluyen:

- Inicialización de Soluciones
- Operador de Cruzamiento
- Operador de Mutación
- Generación de Turnos
- Generación y Validación de Descansos
- Validación de Puntos de Relevo

Para cada uno de ellos, se permite la posibilidad de agregar procesos previo y posterior a su ejecución, a lo que denominaremos *pre* y *post* procesamiento respectivamente. Esto permite incorporar nuevos algoritmos, soluciones y asignaciones que permitan extender, optimizar e incluso restringir las soluciones y procesos deseados.

Para la definición de estos nuevos componentes, bastará con implementar la interfaz definida para el proceso deseado e incorporarlo al EM mediante su configuración. El mecanismo de inclusión, al igual que las extensiones del lenguaje, se basa en el uso de bibliotecas dinámicas (*dll/so*), que permiten la incorporación de nuevos elementos u objetos, en este caso procesos a ser ejecutados e invocados desde lugares definidos. Los detalles sobre este módulo así como una guía para la incorporación de nuevas extensiones pueden encontrarse en el anexo E y anexo G.

```
<Extension>
  <ReliefValidator>
        <pre>
         <libPath>em/ReliefValidatorExtensions.so
         </libPath>
         <objects>
              <obj>RELIEFVALIDATOREXT_A</obj>
              <obj>RELIEFVALIDATOREXT_B</obj>
         </objects>
        </pre>
  </ReliefValidator>
</Extension>
```

Fig. 5.1 Ejemplo de definición de extensiones de pre procesamiento de la validación de puntos de relevo.

5.1. Extensión para Mejora de Libros de Servicio.

Como se detalla desde un principio y se especifica en la función de *fitness*, existen múltiples factores a tener en cuenta a la hora de mejorar soluciones, donde incluso la ponderación de su influencia podrá variar dependiendo del interés particular de cada empresa o persona. Aún así y acorde a la realidad que plantea la empresa en cuestión (COPSA) y sus normativas vigentes, uno de los factores que indica y permite minimizar costos tanto en el uso de recursos materiales como humanos, es la cantidad de turnos que forman al libro de servicios. Lo que debe su causa fundamentalmente a los siguientes factores.

Por un lado, menos turnos implican menor uso de distintos vehículos, lo que tiene un ahorro directo no solo en el combustible, donde probablemente se recurra a menor cantidad viajes del tipo *pull in* y *pull out trips*, sino por el costo asociado al desgaste propio del uso y puesta en marcha de los vehículos. Por otro lado la ventaja de poder disponer de más unidades a la orden para cubrir imprevistos o nuevos requerimientos. Y por último, pero no menos importante, el factor relativo al costo y pago de jornales, donde no se manejan modalidades de pagos por hora trabajada, sino que la asignación de un tripulante a un jornal ya sea por un 1% o 100% de la *duración óptima deseada* del mismo implican el pago de un valor mínimo determinado.

Se buscará, mediante el análisis de una solución, minimizar la cantidad de turnos y vehículos utilizados en ella. Diremos entonces que dos turnos t_1, t_2 son compatibles si la duración del viaje expreso (no estrictamente necesario) desde el destino del último viaje activo de t_1 hasta el origen del primer viaje activo de t_2, es menor a la diferencia entre los horarios de llegada y salida de dichos viajes respectivamente. Lo que permitiría a uno de los vehículos realizar el turno del otro con el posible agregado de un viaje expreso (*deadhead trip*).

Motivados por lo anterior y complementado con el análisis empírico de soluciones (*cantTurnosNoOpt.txt*) las cuales reflejan algunas asignaciones de turnos de corta duración, compatibles entre sí, a distintos vehículos (como se muestra en la Figura 5.1.1.1), es que se implementa, e incluye a modo de ejemplo, una extensión de la generación de turnos, buscando reducir los costos asociados a la misma.

```
...
    Itinerario del vehiculo 21 del garage ubicado en 2.

    Turno   Tipo  Origen Destino Salida Llegada
    12      Activo 2     0       08:40    9:20
    12      Activo 0     2       10:20    11:00
    12      Activo 2     0       11:10    11:50
    13      Activo 0     2       18:00    18:40
    Kilometros Activos: 80
    Kilometros Expresos: 0
    Total: 80
...
    Itinerario del vehiculo 28 del garage ubicado en 2.

    Turno   Tipo  Origen Destino Salida   Llegada
    17      Activo 2     0       05:30    06:10
    17      Activo 0     2       07:50    08:30
    Kilometros Activos: 40
    Kilometros Expresos: 0
    Total: 40
...
```

Fig. 5.1.1.1 Parte de la solución CantTurnosNoOpt.txt que refleja la posible mejora de la solución mediante la unión de dos turnos compatibles. En este caso sin necesidad de un viaje expreso.

Para dicha extensión, se propone y desarrolla la inclusión de un post procesamiento de la solución propuesta por el generador de turnos, que realiza una búsqueda de turnos compatibles realizados por distintos vehículos, de modo de reasignarlos a uno de ellos.

```
shiftOptimizer(Solucion solucion){
  for each turno in solucion
        if (isOptimizable(turno))
              Turno turnoComp =
                obtenerTurnoCompatible(s)
              mergeTurnos(s, turno, turnoComp);
        end
  end;
}
```

Fig. 5.1.1.2 Pseudocódigo del optimizador de turnos para la minimización de la cantidad de turnos.

Para la selección de turnos a optimizar se contemplan los parámetros establecidos para la *duración óptima deseada* y *mínima deseada* de un jornal. Particularmente, se opta por que el vehículo del turno con horario más temprano realice también el turno compatible, liberando al vehículo que lo realizaba, entendiéndose que podrían hacerse análisis más exhaustivos que determinen otras mejoras. A modo de ejemplo, considerando la Figura 5.1.1.1, el vehículo 28 realizaría ambos turnos.

```
<ShiftGenerator>
   <post>
        <libPath>em/ShiftGeneratorOptimizer.so
        </libPath>
        <objects>
              <obj>SHIFTQUANTITYOPTIMIZER</obj>
        </objects>
    </post>
 </ShiftGenerator>
```

Fig. 5.1.1.3 Especificación de configuración para la inclusión post procesamiento del generador de turnos con el proceso SHIFTQUANTITYOPTIMIZER de la biblioteca ShiftGeneratorOptimizer.so

6. Sistema PG42

En la presente sección se describen aspectos generales relevantes al sistema desarrollado como producto. Los detalles del proceso de desarrollo del mismo, así como la especificación de cada uno de los módulos que lo componen se encuentran en los correspondientes anexos (A-E).

6.1. Consideraciones Generales

Uno de los aspectos a tener en cuenta sobre el enfoque del producto desarrollado consta del uso del paradigma de orientación a objetos[DOO] para la implementación de una técnica evolutiva, particularmente para un problema de optimización. Si bien el uso de tal paradigma podría llevar a dificultar uno de los objetivos principales de una implementación para un sistema de este tipo, como lo es la eficiencia en materia de costo computacional, la utilización del paradigma de orientación a objetos aporta múltiples ventajas las cuales justificaron su uso. Dicha metodología aporta grandes ventajas en cuanto a la reusabilidad, legibilidad de código, capacidad de abstracción para el análisis y diseño de problemas, así como su codificación y modularidad.

Considerando que la implementación de una metaheurística, según su definición y objetivos, debería ser independiente del problema que busca resolver, se procura realizar una partición de los conceptos del modelo. Por un lado, el motor de resolución que implementa la metaheurística, en este caso un algoritmo genético, y por otro lado, el problema instanciado. Para facilitar esta abstracción, se recurre al uso de esqueletos algorítmicos, particularmente tomando como base el esqueleto de algoritmo genético provisto por la biblioteca MaLLBa[MaLLBa]. Para esto, se procedió a especificar y extender conceptos para modelar la realidad descrita. Un esqueleto es una pauta algorítmica que implementa un método genérico de resolución, y que debe ser debidamente completado para instanciar y resolver un problema concreto. Los detalles de estos se describen en el anexo D.

6.2. Proceso de Desarrollo

Tratándose de un problema con muchas restricciones, pequeñas variaciones, tanto en la realidad considerada como en los parámetros de configuración, pueden afectar considerablemente los resultados obtenidos a través de las distintas técnicas utilizadas. Esto, sumado a la dificultad implícita que tiene la evaluación y comparación de soluciones para distintos escenarios son factores a tener en cuenta para la obtención de buenos resultados durante la evolución y desarrollo de la solución.

El desarrollo del proyecto se lleva a cabo por etapas respetando un modelo en cascada, donde en cada una de ellas se procuran resolver problemáticas definidas considerando la previa como base. Dicha modalidad surge a raíz de la necesidad de un modelo iterativo e incremental que permita asegurar la validación de las soluciones desarrolladas en cada etapa (dentro de los parámetros de validez definidos). De esta manera se permite concentrar los esfuerzos en la mejora e inclusión de nuevas

funcionalidades o consideraciones a incluir en la etapa actual, logrando así aislar y mitigar los riesgos del impacto que las incorporaciones puedan tener. Por más detalles acerca de la gestión y proceso de desarrollo referirse al anexo A.

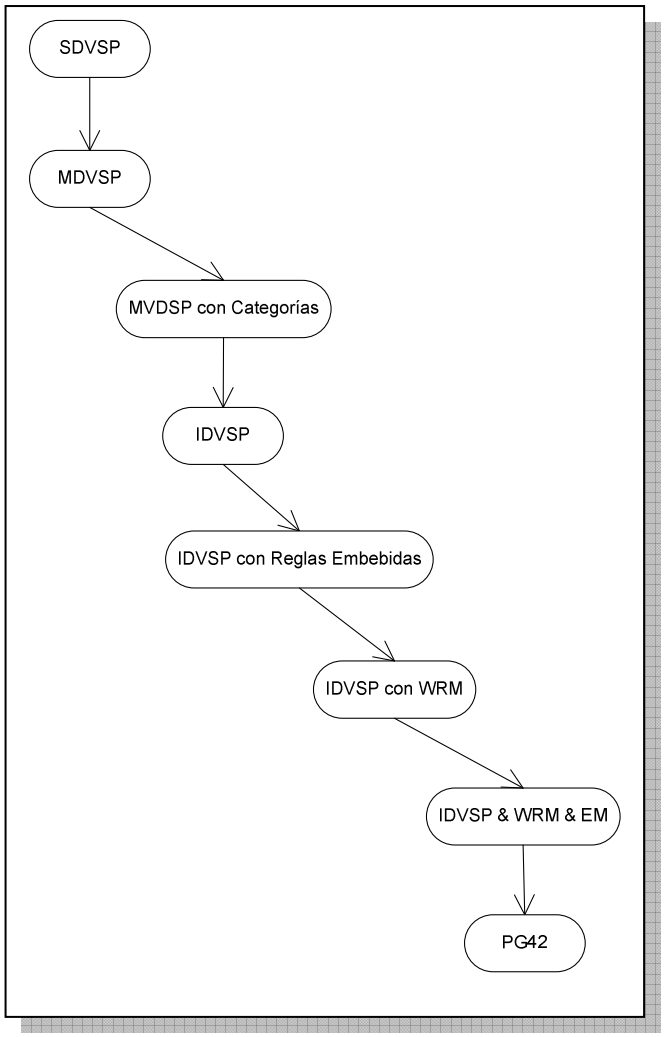

Fig. 6.2.1 Etapas y problemas considerados durante la evolución del proyecto.

6.3. Diseño y Arquitectura

Al tratarse de un sistema que consta de la ejecución de un algoritmo para la resolución de un problema de optimización, donde no existe una interacción constante con el usuario (más que la configuración de datos de entrada y despliegue de resultados), ni persistencia de entidades, es que su diseño consta principalmente de lo que sería la capa lógica de un modelo *multi-tier*. Los distintos componentes y procesos del sistema son agrupados en módulos (paquetes) de acuerdo a su funcionalidad, interactuando entre sí para llevar a cabo la ejecución.

En las secciones a continuación se presentan aspectos generales sobre el diseño, estructura y componentes de la solución desarrollada.

6.3.1. Diagrama de Paquetes

La Figura 6.3.1 muestra el diagrama de paquetes del sistema y las interfaces definidas para la utilización de los módulos WRM y EM. Se presenta a la biblioteca MaLLBa como un subsistema dada la extensión de componentes e implementación de interfaces realizadas en el módulo GAM.

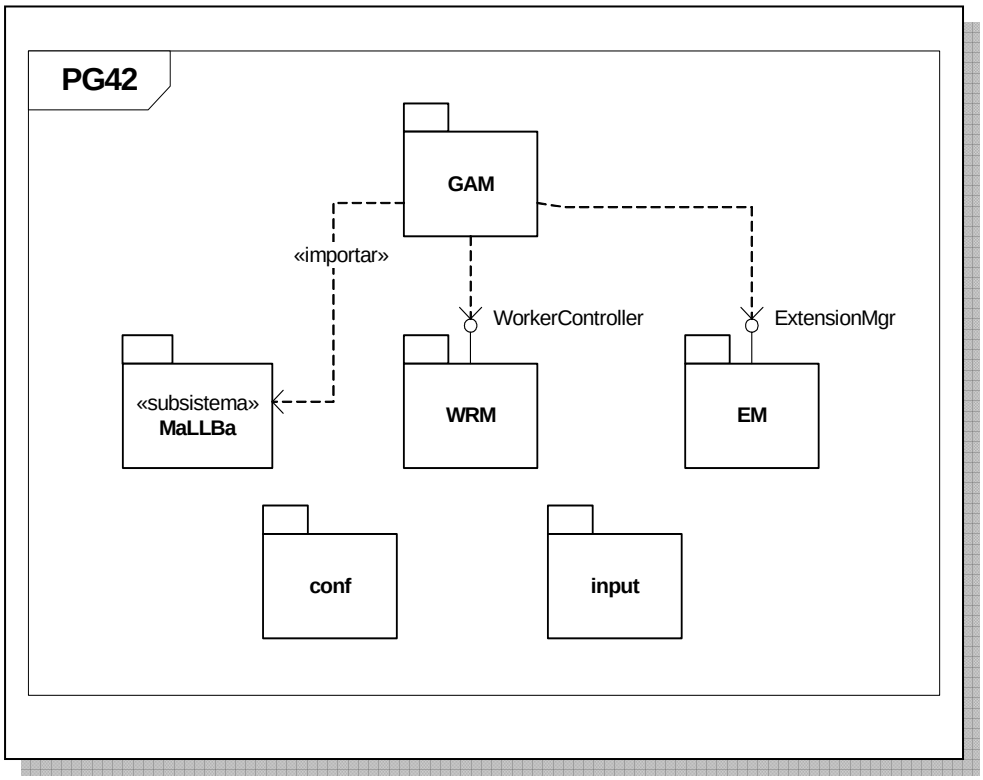

Fig. 6.3.1. Diagrama de Paquetes del Sistema PG42.

6.3.2. Biblioteca MaLLBa

Considerando el problema a resolver y posibles soluciones, se propone la utilización de una biblioteca reconocida en el ámbito académico como ser MaLLBa[MALLBA2]. Entre los motivos que impulsan el uso de la misma se encuentran: experiencias satisfactorias (propias y ajenas) en cuanto a su uso y resultados obtenidos que han probado su eficacia y eficiencia[MALLVRP]. Así como también el hecho de ser una biblioteca libre y abierta con un diseño simple y posibilidades de modificación, adaptación, extensión y modularización de cualquiera de sus componentes. Además MaLLBa provee compatibilidad con MPICH[MPICH], una implementación de MPI[MPI] que anticipa la posibilidad de computación paralela. Dada la importancia de esta biblioteca por detalles sobre la misma referirse al anexo D.

6.3.3. Módulo de Algoritmo Genético (GAM)

Su funcionalidad principal consta de la búsqueda y generación de soluciones factibles respetando las restricciones de ordenamiento propias del problema, así como la integración de las normativas laborales mediante la integración del módulo de reglas laborales (WRM). Siendo el módulo que contiene al motor de ejecución (algoritmo genético), es quien contiene el método principal (ejecutable), instancia del problema especificada en el los paquetes *Input* y *Conf* correspondiente a los datos de entrada y configuración respectivamente; e integra al resto de los módulos de acuerdo a los requerimientos de extensibilidad y bajo acoplamiento establecidos en el anexo B.

La Figura 6.3.3.1 muestra un diagrama de clases de diseño del módulo donde se muestra la estructura implementada para el esqueleto y motor de ejecución del algoritmo. La especificación ampliada de cada uno de los elementos así como demás componentes se encuentra en el anexo B.

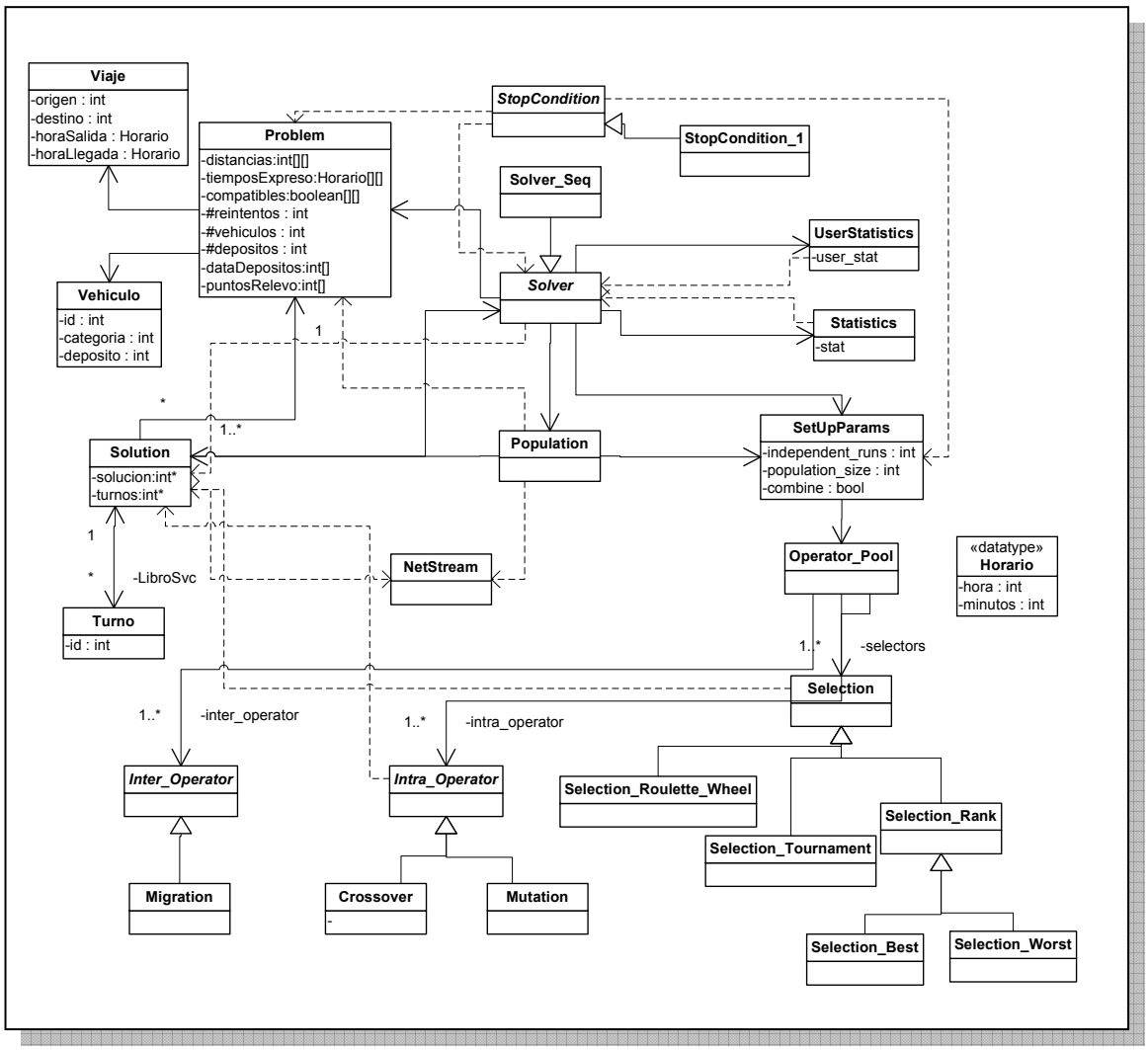

Fig. 6.3.3.1 Diagrama de Clases de Diseño de GAM

6.3.4. Módulo de Reglas Laborales (WRM)

Este módulo tiene dos objetivos fundamentales. Uno de ellos es encargarse de toda la gestión de las normativas laborales. Esto es, definición, creación y validación de las mismas. El otro objetivo que tiene a cargo es el de la generación de turnos. Para llevar a cabo esta tarea, utiliza las reglas laborales, ya que los mismos deben cumplir con todas las reglas que se hayan definido.

La Figura 6.3.4.1 muestra un diagrama de clases de diseño del módulo. La especificación ampliada de cada uno de los elementos así como demás componentes se encuentra en el anexo C.

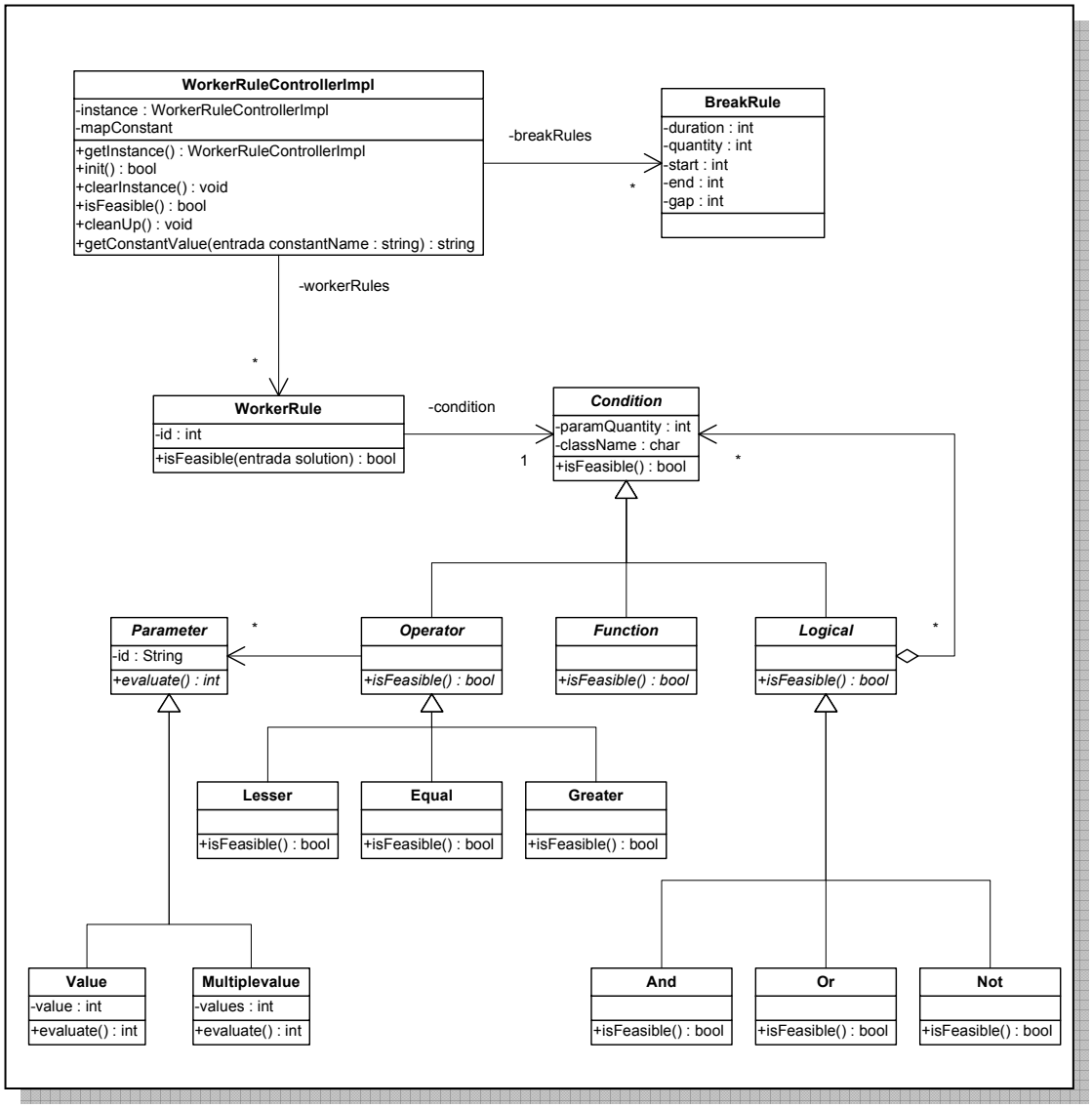

Fig. 6.3.4.1 Diagrama de Clases de Diseño de WRM.

6.3.5. Módulo de Extensiones (EM)

El objetivo fundamental de este módulo es proveer la posibilidad de extender e incluir nuevas técnicas y estrategias, es decir, de poder integrar procesos que permitan pre-procesar o post-procesar los resultados o estrategias del algoritmo de búsqueda, optimización y generación de turnos. Esto tendrá como fin obtener mejores resultados, soluciones más ajustadas a las necesidades e intereses, sin tener que modificar el núcleo (*core*) de ambos módulos.

La Figura 6.3.5.1 muestra un diagrama de clases de diseño del módulo. La especificación ampliada de cada uno de los elementos así como demás componentes se encuentra en el anexo E.

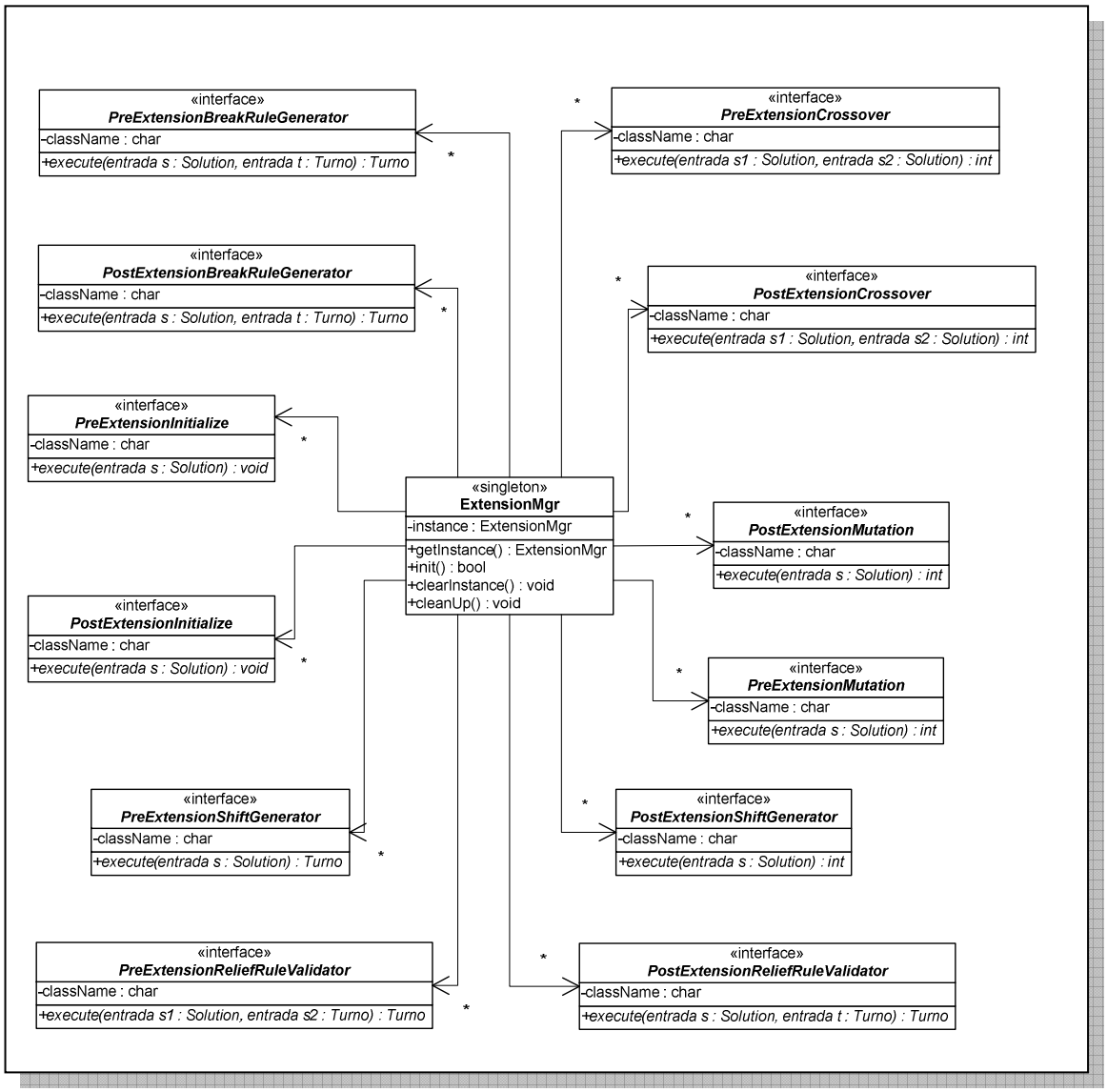

Fig. 6.3.5.1 Diagrama de Clases de Diseño de EM.

7. Experimentos, Resultados y su Análisis

En esta sección se presentan dos experimentos significativos realizados para la evaluación y análisis de la solución desarrollada. Las distintas pruebas pretenderán evaluar múltiples aspectos de la solución, entre ellos, el comportamiento de las estrategias desarrolladas, módulos y métodos implementados, así como también resultados obtenidos. Para cada caso, es decir una instancia del problema que modelará una realidad determinada, se evaluarán distintos escenarios. Cada escenario representará la consideración de una problemática particular para ese caso, por ejemplo distintas restricciones o reglas laborales a considerar para realizar la asignación de flota y tripulación. Para cada caso, se realizarán análisis comparativos entre los distintos escenarios de modo de poder evaluar la calidad de soluciones obtenidas, en términos de *fitness* y sus componentes, tiempos de ejecución, y aspectos de interés de las mismas, como ser restricciones consideradas en cada escenario (ordenamiento de flota, con inclusión de reglas laborales embebidas, y con reglas definidas por el usuario).

Para la ejecución de las pruebas se utilizan los valores de los parámetros obtenidos en la sección 3.6.

Todas las pruebas fueron realizadas en una máquina virtual VMware Server Console[VMWARE] 1.0.5 con 2 procesadores y 512 Mb de memoria asignados corriendo sobre un equipo Dell Latitude E6400[DELL] con procesadores Intel[INTEL] Core 2 Duo de 2,26 GHz y 4Gb de memoria RAM.

7.1. Escenarios

7.1.1. Ordenamiento de Flota

La realización de este escenario permitirá observar la *performance* del algoritmo de ordenamiento de la flota, teniendo como objetivo analizar la calidad de las soluciones obtenidas y poder establecer una cota en los factores de interés que componen el costo (cantidad de turnos, vehículos, kilómetros expresos, etc.), para casos más restringidos. No se imponen aquí fuertes restricciones en cuanto a descansos de la tripulación, u otras reglas laborales, sino que se toman en cuenta únicamente las restricciones de ordenamiento de flota y duración óptima de turnos para su generación.

7.1.2. Con Reglas Laborales Embebidas

Este escenario permitirá evaluar cuánto varía la *performance* con respecto a la ejecución de ordenamiento de flota. Las restricciones planteadas para este caso implican que cada turno tenga una duración inferior a las 8 horas, y que tengan tres descansos de quince minutos o uno de media hora, como lo indica la normativa vigente en la empresa COPSA al momento de realizarse este estudio.

7.1.3. Con Restricciones Embebidas y Definidas por el Usuario

La tercer y última prueba permitirá profundizar, aún más, en el chequeo de la *performance*, teniendo en cuenta que aquí se suma el costo de ejecución del módulo de reglas laborales para la validación de las reglas definidas. Para ello, agregaremos una regla definida por el usuario, además de las reglas embebidas, ya mencionadas en el punto anterior. La restricción definida por el usuario implicará que todos los turnos tengan una duración superior a 3 horas o permitir la existencia de turnos de menor duración, pero exigiéndole a estos, que tengan una alta relación (*r*) de kilómetros activos con respecto a los expresos *(kmActivos/kmTotales=r*).

7.2. Caso de Estudio 1: *Héctor*

Uno de los casos utilizados para experimentar y evaluar, es un caso que ha sido de referencia en proyectos realizados en el INCO, denominado *Caso de Héctor*. Este consta del modelado de una realidad abstracta y significativa en cuanto a representación de un caso real, creada por el grupo de IO especialmente para realizar evaluaciones y pruebas funcionales de sus implementaciones. Este caso contiene 66 viajes que se realizan entre 3 ciudades, con una flota distribuida en 3 depósitos.

Para todas las pruebas se intentarán mejorar los resultados utilizando el proceso de optimización desarrollado para minimizar cantidad de turnos a través de la unión de pares compatibles, el cuál fue descrito en la sección 5.1.

7.2.1. Resultados

En la tabla de la Figura 7.1.2.1, se presentan los resultados de las pruebas sin utilizar el optimizador de turnos.

	Costo	Tiempo (seg)	KE (Km)	#Vehículos	#Turnos	#Horas (hh:mm)	CHDM (hh:mm)
Ordenamiento de Flota	40800	55	180	5	11	57:30	2:10
Con Reglas Embebidas	46860	70	180	6	12	63:00	4:45
Con Reglas Embebidas y Definidas por el Usuario por el usuario	51050	79	220	6	12	64:35	4:45

Fig. 7.1.2.1. Resultados obtenidos para caso *Hector*

En la tabla anterior, la columna *KE*, representa los kilómetros expresos, mientras que la columna *CHDM*, representa la cantidad de horas por debajo del jornal mínimo. En las pruebas se fijó el jornal mínimo en 4 horas, de manera que cada turno que tenga una duración inferior a 4 horas es penalizado en la función de *fitness*.

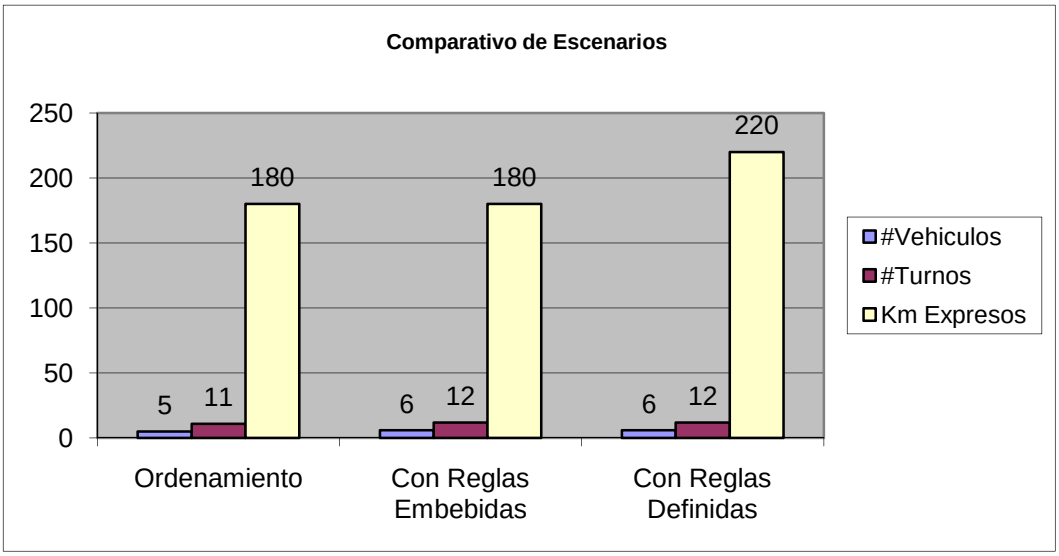

Fig. 7.1.2.2. Gráfico comparativo de los principales factores de los distintos escenarios.

En cuanto al escenario de ordenamiento (sin restricciones), el sistema PG42 obtiene una solución que puede contemplarse como referencia, representando una posible cota para el resto de los casos, ya que estos últimos, al encontrarse más restringidos, podrán requerir mayores costos, que eventualmente podrán aproximarse a la solución sin restricciones. Particularmente sobre este escenario, se cubren los 66 viajes utilizando un 55% de la flota disponible, con un costo total en kilometraje de 1700km compuestos por: 1620km activos y 180km expresos (11.1% del total de kilómetros).

Por otra parte, se evalúa el uso de la extensión de minimización de cantidad de turnos, obteniendo los siguientes resultados.

	Costo	Tiempo (seg)	KE (Km)	#Vehículos	#Turnos	#Horas (hh:mm)	CHDM (hh:mm)
Ordenamiento de Flota	37900	59	180	5	10	57:30	0:00
Con Reglas Integradas	42000	85	180	5	11	62:30	2:40
Con Reglas Definidas por Usuario	44060	109	180	6	11	63:00	2:40

Fig. 7.1.2.3. Resultados obtenidos para caso *Hector* con optimizador de cantidad de turnos.

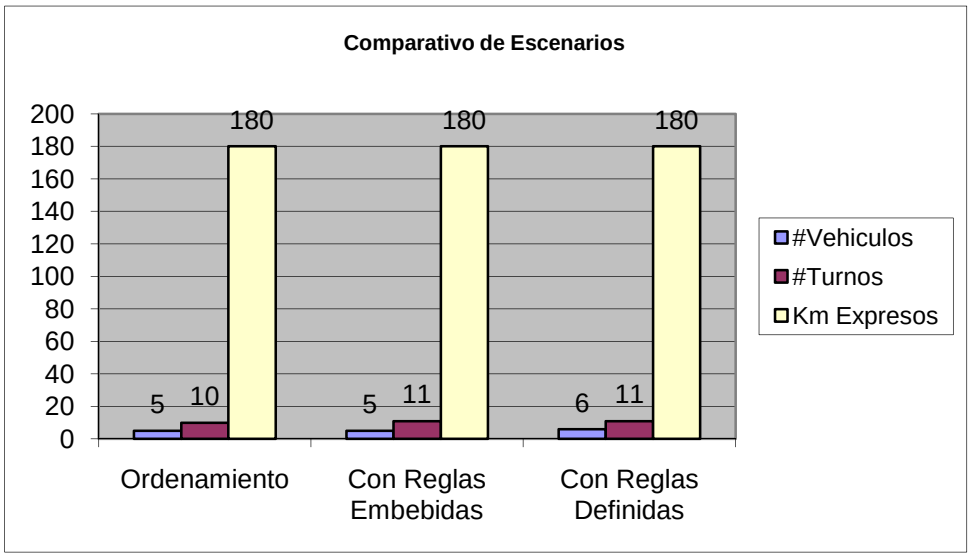

Fig. 7.1.2.4. Gráfico comparativo de los principales factores de los distintos escenarios ejecutados con optimizador.

Como vemos, en todos los escenarios, se obtienen mejoras en las soluciones obtenidas, en comparación a las generadas sin la extensión. Además, se puede apreciar que en los escenarios con restricciones embebidas y definidas, existen incrementos acordes (relativos al de ordenamiento) en los tiempos de respuesta, lo cual es razonable debido al aumento necesario en los costos de ejecución, particularmente sobre el módulo de reglas laborales (WRM) para la validación de individuos.

Acerca de las cualidades de las soluciones obtenidas, cabe notar que la solución con restricciones definidas por el usuario requiere al menos 6 vehículos. A nivel de toma de decisiones, la empresa, podría utilizar esta funcionalidad, para verificar la factibilidad de aplicar cambios en la reglamentación laboral.

7.2.1.1. Comparativo con Sistema IO

Otro de los comparativos valederos, corresponde con la solución obtenida por el grupo de investigación operativa utilizando el sistema desarrollado en el Proyecto

"Ordenamiento de Vehículos 2008-v1", particularmente para el escenario con reglas embebidas. Pese a que la solución desarrollada por el grupo contempla asignaciones de cubrimiento parcial con respecto a la totalidad de viajes a ser cubiertos, se han podido obtener buenos resultados con 93.9% de cobertura (IO-*solucionMayorCubrimiento.txt)* por lo que ameritan ser comparados contra las obtenidas por el sistema PG42 el cual trabaja con cobertura total. Para este escenario se consideran las reglas laborales propuestas por el grupo de IO, considerando 50 minutos de descanso distribuidos en intervalos de descanso con duración superior a 10 minutos y múltiplo de 5 minutos.

	# Viajes	KE (Km)	#Vehículos	#Turnos	#Horas
IO Mayor Cubrimiento	62	0	6	14	123:50
PG42 Con Reglas Embebidas – Escenario IO	66	220	5	12	76:55

Fig. 7.1.4.4. Comparativo de caso *Hector* - Reglas Embebidas – Escenario IO, con optimizador de cantidad de turnos, contra la solución de mayor cubrimiento del Dpto. de Investigación Operativa.

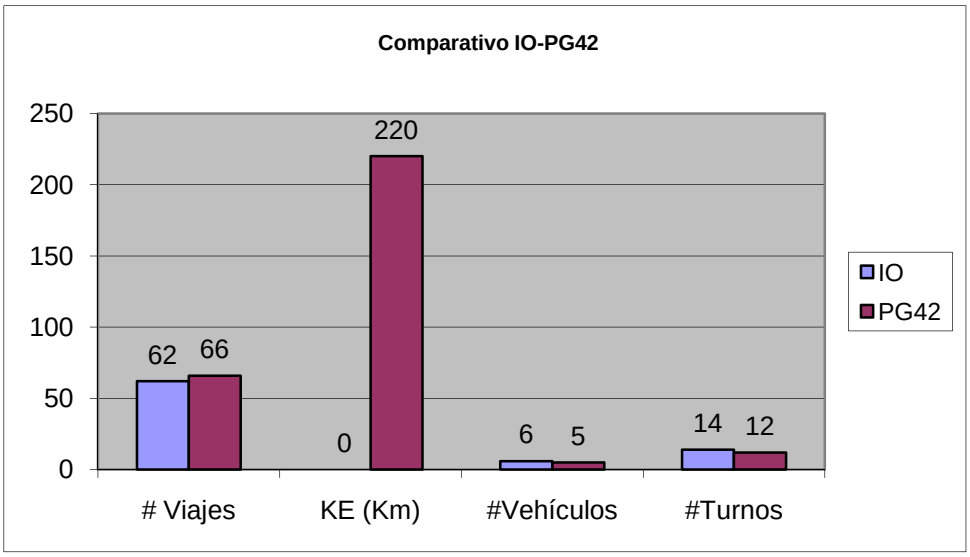

Fig. 7.1.4.5. Comparativo entre los resultados obtenidos por los sistemas de IO y PG42.

Debe tomarse en cuenta aquí, que se está comparando contra una solución donde existen costos omitidos en, por lo menos, uno de los factores presentados, ya que para cubrir los viajes no asignados, deberán necesariamente incrementarse los costos en cuanto a kilómetros expresos, vehículos utilizados o cantidad de turnos. La solución obtenida por el sistema PG42 refleja como desventaja la necesidad de viajes expresos

(*deadhead trips*) totalizando un costo de 220km. Las distancias involucradas en este caso promedian unos 30km entre nodos, por lo que el costo estaría asociado a unos 7 viajes expresos que podrán ser invertidos tanto en *deadhead trips* intermedios a viajes activos (*active trips)*, como también en *pull in* o *pull out trips*. Pese al kilometraje adicional, el cual representa un 12% del kilometraje total recorrido por la flota asignada (1800km), la solución presenta grandes ventajas que justifican tal costo. En primer lugar se resuelve el ordenamiento utilizando un vehículo menos (16.6%), lo que refleja el buen comportamiento de las estrategias de reutilización de vehículos procuradas durante la etapa de ordenamiento, incluyendo las distintas técnicas de inicialización secuencial y por entronques. También la solución obtenida refleja un buen comportamiento relativo a la eficacia de las técnicas de minimización de jornales laborales (turnos-*duties*), obteniendo un total de 12 turnos, lo cual implica menor costo en horas hombre con un ahorro de 2 jornales (14.3%). Según la ponderación de estos costos, la empresa podrá determinar cuál de las opciones es más conveniente.

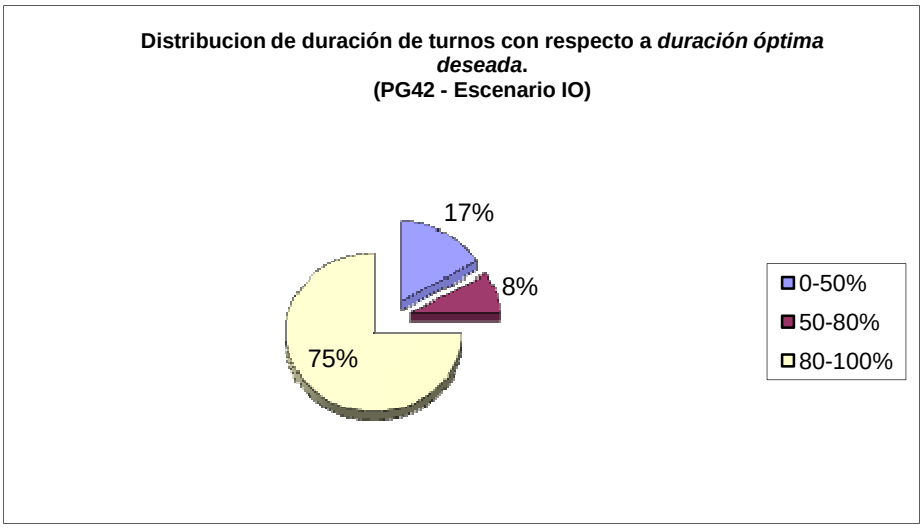

Fig. 7.1.4.6. Gráfico de distribución de la duración de los turnos respecto a duración óptima (8hs).

7.3. Caso de Estudio 2: *COPSA*

El caso COPSA[COPSA] es un caso real de una empresa operando en el medio local. En este caso se pretende evaluar el comportamiento del sistema desarrollado frente a una instancia de alta complejidad incluyendo un alto volumen de datos (viajes, nodos) y procesamiento de los mismos.

Este caso contiene más de mil viajes (1069), y 182 nodos entre ciudades y depósitos. Se cuenta con una flota heterogénea de 335 vehículos, distinguida entre 2 categorías y distribuída entre 8 depósitos promediando un total de 41 vehículos por depósito, teniendo un mínimo de 17 vehículos y máximo de 83 vehículos en dos de sus depósitos.

Para la especificación del problema, se utilizan datos provistos por el Dpto. de Investigación Operativa, debiendo resolver la falta de unas 3000 entradas (cerca del 10% de las mismas) en la tabla de distancias y tiempos entre nodos, definiéndose valores promedio entre ellos.

7.3.1. Resultados

Considerando los costos que implica el caso de COPSA y aspectos que se presentaran a continuación, se consideran como relevantes las pruebas utilizando el optimizador de cantidad de turnos (*SHIFTQUANTITYOPTIMIZER*).

	Costo	Tiempo (hr)	KE (Km)	#Vehículos	#Turnos	#Horas
Ordenamiento de Flota Con reglas Embebidas	132998e+07	5:08	10010	115	300	1802
	181543e+07	9:43	29664	183	234	1707

Fig. 7.2.1.1. Tabla de valores obtenidos para los distintos escenarios del caso COPSA.

Entre las principales observaciones a realizar acerca del comportamiento del sistema, particularmente con respecto al caso con restricciones, se encuentran las relativas a la inicialización de soluciones. Se observa que un alto porcentaje del tiempo de ejecución del algoritmo (40% aprox.) corresponde a la inicialización. Dicho costo puede atribuirse a la dificultad de obtención de soluciones factibles para formar la población inicial, principalmente a través de las estrategias de inicialización secuencial y por entronques. Estas estrategias buscan asignar un mismo vehículo a viajes repetidamente, dejando así poca flexibilidad para la incorporación de nuevos eventos, principalmente descansos.

La no incorporación de reglas laborales o restricciones a los algoritmos de generación de soluciones iniciales tiene como desventaja, una alta probabilidad de descarte de soluciones que pudieran aportar buenas propiedades, debido a la existencia de alguna (al menos una) incompatibilidad correspondiente a las restricciones definidas. Entre las alternativas posibles para disminuir dichos costos se podría, en principio, ejecutar la inicialización de manera independiente, y reutilizar las soluciones generadas en otras ejecuciones independientes. También, considerando lo anterior, una de las mejoras propuestas consiste en la implementación de un generador de soluciones iniciales que mejore estos aspectos, como se detalla en la sección 9.

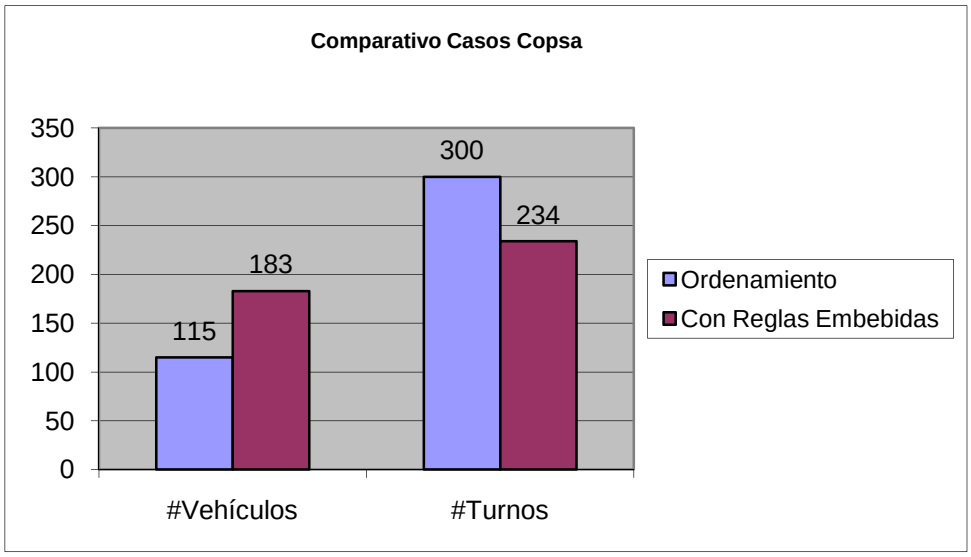

Fig. 7.2.1.2. Comparativo de cantidad de vehículos utilizados y cantidad de turnos requeridos para los distintos escenarios del caso COPSA,

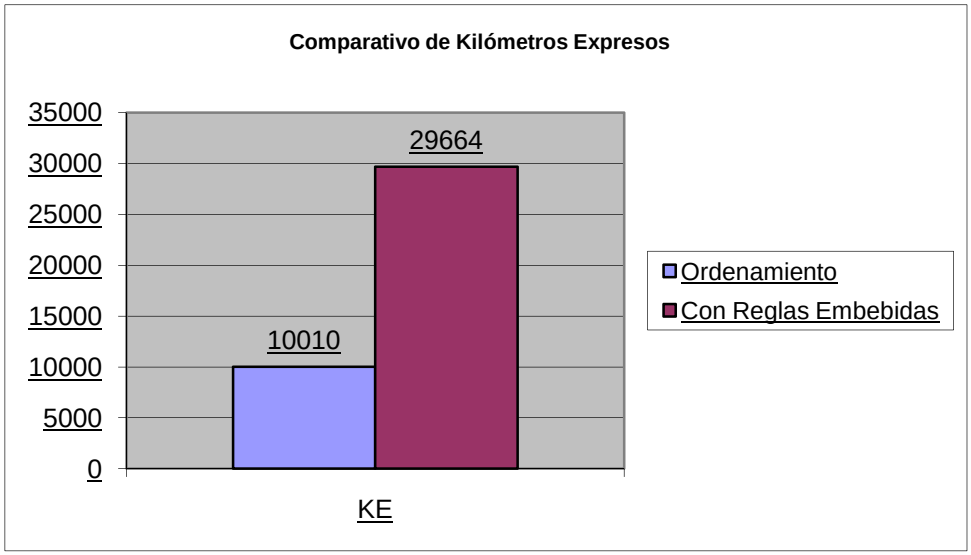

Fig. 7.2.1.3. Comparativo de cantidad de kilómetros expresos recorridos en viajes (*deadhead trips*) para los distintos escenarios del caso COPSA

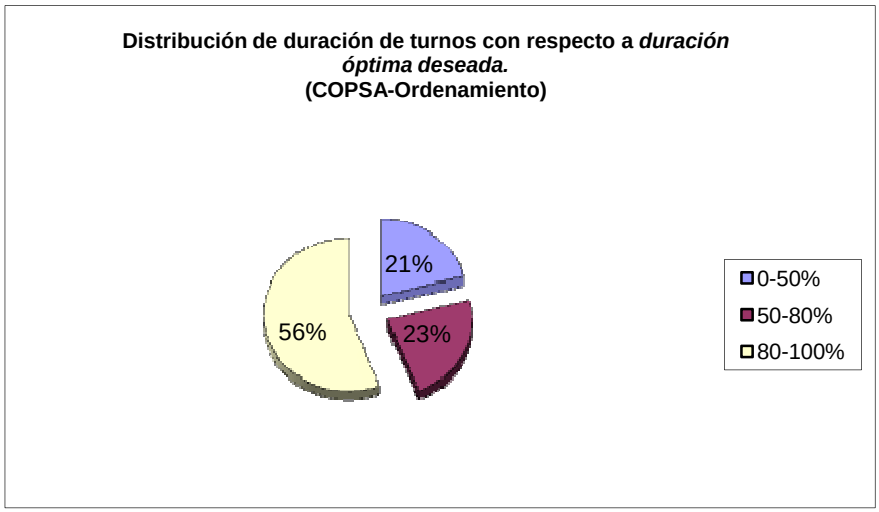

Fig. 7.2.1.4. Distribución de la duración de turnos respecto a la Duración Óptima – Escenario COPSA Sin Restricciones. La duración óptima es la especificada en las reglas actuales de (8hs).

El escenario de ordenamiento (sin restricciones), muestra que un 79% de los turnos se encuentran por encima del jornal mínimo deseado, especificado en 4hs, lo cual puede considerarse aceptable con la penalización establecida para jornales inferiores a dicho valor.

Cabe notar que la cantidad de turnos requerida en el escenario menos restringido, es superior al escenario con restricciones. A priori, se podría pensar que el escenario con restricciones debería fijar máximos en el uso de recursos para escenarios menos restringidos. Es decir, que una solución de un escenario restringido, es contenida por el conjunto de soluciones del escenario menos restringido. Es cierto que en algunos casos, considerar dichos valores de ese modo podría ser de utilidad, ya que a partir de ellos podría compararse y evaluarse el comportamiento de técnicas y algoritmos. Sin embargo en este experimento, el sistema encontró como mejor solución, una en la cual prevalecen los beneficios aportados por otros factores (km expresos, cantidad de vehículos, etc.) sobre la minimización en la cantidad de turnos. Es decir, que aún con técnicas de minimización de turnos y conociendo una solución que posee menos turnos, se obtuvo una exploración del espacio de soluciones y diversidad suficiente en la población, para obtener como solución, un individuo que no minimiza la cantidad de turnos (ya que bastaría con considerar la solución del escenario con restricciones para tener una con menor cantidad). Particularmente, y comparando ambos escenarios, se entiende que tal selección puede ser realizada debido a que el incremento en 66 turnos (28%), no refleja su impacto en el costo de horas hombre. Esto se basa en que la diferencia entre las soluciones con respecto a este factor es de 95hs, que implicarían 12 turnos adicionales aproximadamente si tuvieran una duración de 8hs. Esto se refleja, en los resultados de distribución de turnos respecto a la duración óptima, donde en el escenario con restricciones embebidas existe mayor aprovechamiento de cada turno, obteniendo así menor cantidad. Por otro lado, en el escenario sin restricciones, se opta por absorber dicho costo con el objetivo de minimizar el kilometraje en un 33.7% (19654km) así como también disminuir la flota utilizada en un 37.1% (68 vehículos).

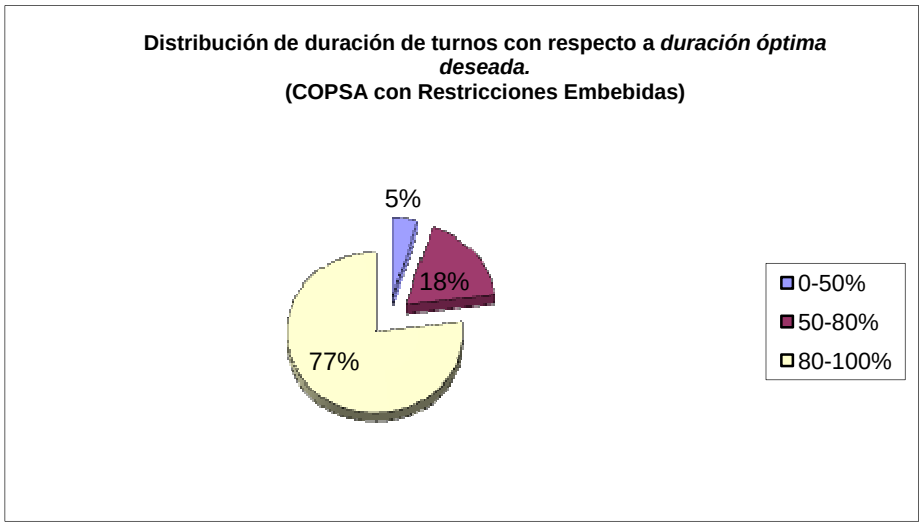

Fig. 7.2.1.5. Distribución de la duración de turnos respecto a la *duración óptima deseada*. Escenario COPSA con Restricciones Integradas. La duración óptima es la especificada en las reglas actuales (8hs).

Para el escenario con restricciones embebidas, pese al kilometraje expreso obtenido, se refleja una buena distribución en cuanto al aprovechamiento de los jornales, teniendo en más de tres cuartos (77%) de los turnos una duración superior al 80% de la *duración óptima deseada* especificada en 8 horas.

Entre los aspectos destacables de la ejecución cabe mencionar que el sistema permite la obtención de soluciones factibles con cobertura total de viajes. También en todas las pruebas se observa la influencia de la técnica de minimización de uso de la flota, utilizando solamente entre un 34% y 45% de la flota disponible.

Entre los aspectos a mejorar sobre la calidad de las soluciones obtenidas se pueden apreciar, que con el objetivo de minimizar la cantidad de turnos y vehículos utilizados, se incrementan significativamente los viajes expresos realizados. Uno de los factores sobre los cuales recae este comportamiento, se debe al uso de la técnica empleada en la extensión de mejora de cantidad de turnos, el cual en principio no realiza un análisis significativo para evaluar el impacto del agregado de viajes expresos de larga duración para la unión de turnos compatibles. Es decir, el optimizador centra su esfuerzo en la minimización de turnos y vehículos, siendo esto prioritario ante el aumento de costos incurridos por kilometraje expreso. Dependiendo de la instancia, éste comportamiento podría empeorar la relación entre viajes activos y expresos en un turno. Por ejemplo, con una inicialización aleatoria, donde gran parte de la población inicial consta de turnos de corta duración, la extensión intentará unir la mayor cantidad de estos, agregando viajes expresos, ocasionando así un incremento en el coeficiente *kmExpresos/kmActivos*, el cual se espera sea lo menor posible.

7.3.1.1. Comparativo IO

Análogamente al caso anterior, existe una solución generada por el Departamento de Investigación Operativa con la cual podremos establecer conexiones. Cabe aclarar que existen diferencias sustanciales en cuanto al enfoque de la solución propuesta, ya que como se mencionó en la sección 7.1 existen diferencias en el proceso de generación y factibilidad de las soluciones. Las mismas, en el caso de IO, pueden contener un mismo viaje asignado a más de un turno, y no consideran cobertura total de viajes, lo cual representa una diferencia circunstancial en cuanto al formato de las soluciones y factibilidad de las mismas respecto al sistema PG42.

Aún así, se realizará un comparativo a modo de establecer tanto aspectos positivos como negativos de las soluciones generadas por el sistema, incluyendo la asignación desarrollada manualmente por los encargados de la planificación de la empresa en cuestión. Vale aclarar que esta solución, es fruto de años de experiencia, ajustes incrementales, y meses de trabajo para su desarrollo.

Se plantean, en la Figura 7.2.1.1.1, los principales aspectos de las soluciones y los valores que se han obtenido en cada uno de los sistemas.

	Km Expresos	#Viajes	#Vehículos	#Turnos	Tiempo Ejecución (hh:mm)
PG42	29664	1069	183	234	9:43
IO	4012	997	233	466	17:00
COPSA	565	1069	240	480	N/A

Fig. 7.2.1.1.1. Tabla de valores obtenidos para el escenario de COPSA, utilizando los distintos sistemas y métodos.

Por un lado, puede destacarse que la solución obtenida utilizando PG42 tiene una cantidad de kilómetros expresos notoriamente superior, sin embargo, presenta ventajas con respecto a los demás factores considerados. Una de las desventajas que sufre la solución de IO corresponde a la cobertura parcial de los viajes activos (93.3%), lo cual representa un total de 72 viajes no asignados donde se omiten sus costos relacionados a viajes expresos, vehículos y turnos adicionales para cubrirlos. Por otro lado, la solución del departamento de transporte de COPSA, efectivamente cubre todos los viajes, con la desventaja principal del tiempo de generación y esfuerzo requerido el cual se encuentra en el orden de semanas de trabajo.

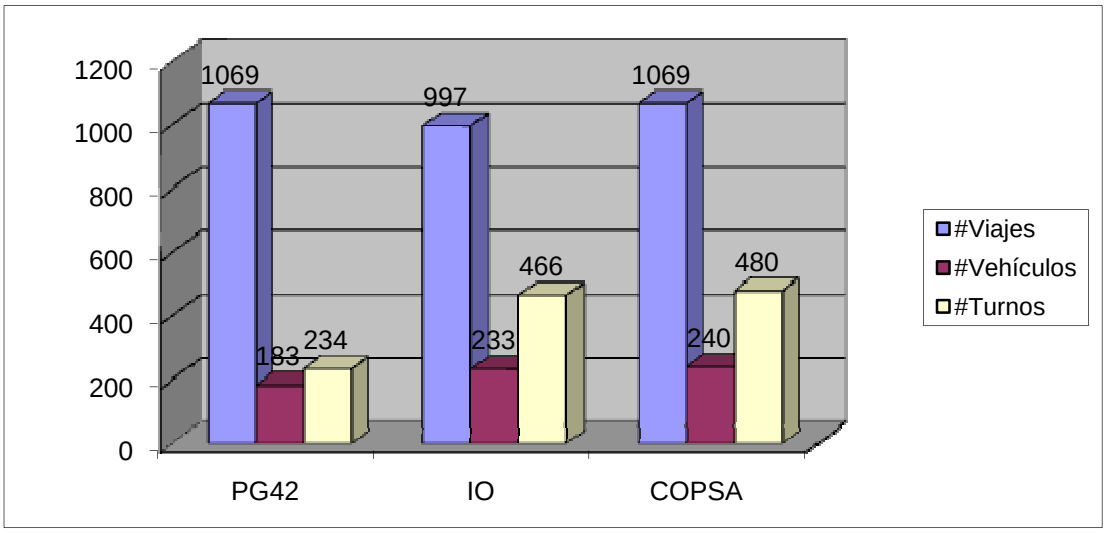

Fig. 7.2.1.1.2. Gráfico comparativo de factores clave de las soluciones obtenidas por los distintos sistemas.

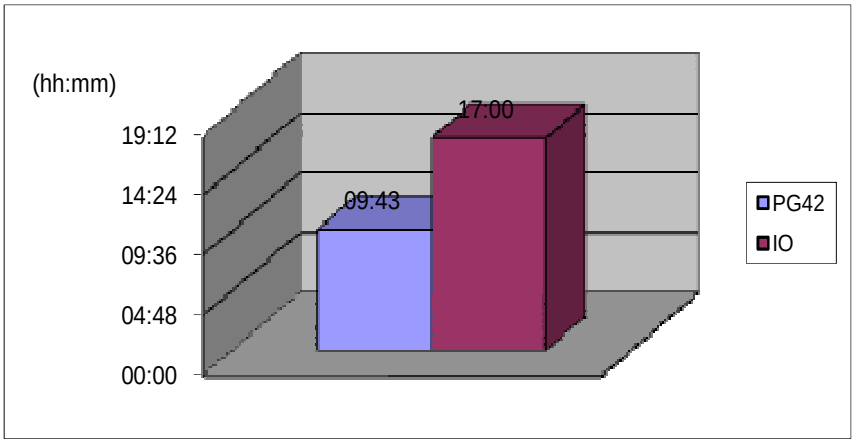

Fig. 7.2.1.1.3. Gráfico comparativo de tiempo de ejecución de las soluciones obtenidas por los distintos sistemas. No se incluye la solución de departamento de transporte de COPSA debido a que su elevado costo no aplica a la comparación con sistemas automatizados.

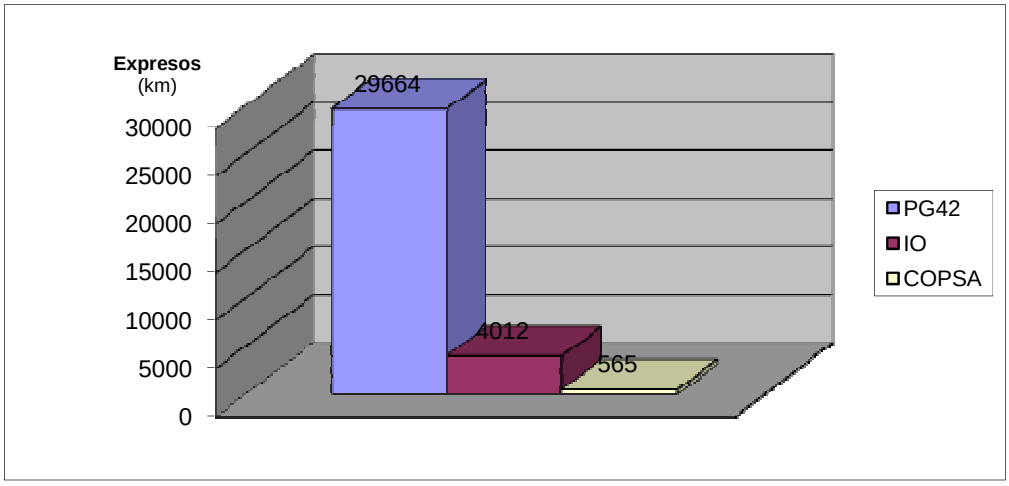

Fig. 7.2.1.1.4. Gráfico comparativo de kilómetros invertidos en viajes expresos (*deadhead trips*) de las soluciones obtenidas por los distintos sistemas.

Al igual que en el caso anterior, para el sistema PG42, y en comparación con las otras soluciones, se observa la influencia de la minimización de la cantidad de turnos en las soluciones obtenidas. Como se menciónó en la sección 7.3.1, ésta extensión repercute en el incremento de los kilómetros expresos necesarios, donde en este caso, y dado el tamaño del problema, se observa un incremento importante relativo a las demás soluciones. Pese a que la función de *fitness* intenta ponderar de manera equilibrada la cantidad de turnos y vehículos utilizadas con respecto a los kilómetros, es evidente que en este caso, dicho objetivo es descompensado por el de la extensión de reducción de cantidad de turnos, situación que como se mencionó previamente, y de interés particularmente en este caso, podría mitigarse mediante alguna estrategia que limite la incorporación de nuevos viajes expresos. De todos modos, se obtienen buenos resultados en cuanto al resto de los objetivos en comparación con las demás soluciones, obteniendo menores costos en todos los demás factores. En materia de vehículos, la solución de PG42 utiliza 78% y 76% con respecto a la de IO y COPSA respectivamente. Y en cuanto a la cantidad de turnos, los beneficios son mayores aún, realizando la asignación de todos los viajes utilizando un 50% de los turnos aproximadamente con respecto a las demás.

7.4. Consideraciones Generales del Sistema

A modo de conclusión, el sistema muestra un comportamiento coherente frente a las distintas pruebas realizadas. En materia de tiempos de ejecución, se pueden observar incrementos acordes entre los casos y complejidad de los distintos escenarios.

Acerca del comportamiento del sistema y el cumplimiento de sus objetivos, en uno de los casos, particularmente el utilizado para las pruebas de verificación (*Hector*), se obtienen buenos resultados, reflejándose valores acordes en todos los factores que componen la función de *fitness* como se describen en la sección 7.2. Para el segundo caso (COPSA), se puede apreciar que muchas de las técnicas aplicadas conducen a buenos resultados, sin embargo, también se observa que el no preveer ciertos comportamientos o limitaciones en el uso de técnicas auxiliares, como lo es la mejora de la cantidad de turnos, pueden conducir a poblaciones que descuiden los costos que no

fueron tenidos en cuenta, como ocurre con los kilómetros expresos en este caso. De todos modos, y previendo este tipo de comportamiento, el cual usualmente tiene una alta complejidad para su anticipación, debido a que requieren la simulación o análisis teórico de la confluencia de múltiples estrategias y algoritmos (posiblemente aleatorias), es que se procura el desarrollo de un sistema flexible. La incorporación de técnicas de mejora de soluciones como extensiones a las incluidas por defecto, muestra tener buenos resultados en los casos, aunque deberá tenerse especial consideración de no impactar negativamente a los demás factores cuando se intenta mejorar uno de ellos, problema inherente de una función multiobjetivo.

8. Conclusiones

Afortunadamente, la experiencia con el análisis y aplicación de metaheurísticas, particularmente de algoritmos genéticos, muestra un comportamiento que intenta resolver un problema de naturaleza compleja, con un espacio de soluciones restringido a un conjunto heterogéneo y complejo de factores e intereses. Aunque en muchos casos es posible que las soluciones obtenidas no sean óptimas, lo cual se considera un problema inherente de la utilización de metaheurísticas, se buscó desarrollar estrategias y mecanismos como lo plantean los objetivos de este trabajo, de manera de poder adaptar y mitigar dicha desventaja mediante el análisis, diseño, abstracción, parametrización y generación de un sistema que pretenda ser eficaz, eficiente, genérico y extensible. De este modo se permite que, sobre un problema que no ofrezca los resultados esperados, se puedan realizar e integrar mejoras, como se ha procurado comprobar.

En función de lo anterior, se considera que la aplicación de dicha metaheurística provee un mecanismo definido y flexible en cuanto a la técnica y enfoque para enfrentar una problemática compleja como la planteada, pero tanto las técnicas y estrategias particulares que la conforman podrán depender considerablemente del escenario o instancia particular si se esperan obtener buenos resultados. Es así, que se considera como prioridad en el desarrollo de soluciones, incluyendo la aquí provista, tener un alto grado de parametrización y extensibilidad, que permitan la adaptación e incorporación de conocimiento del problema, con el objetivo de mejorar los resultados y realizar modificaciones significativas. Entre ellas, cambios que podrán partir desde su arquitectura (por ejemplo ejecución distribuida o extensión del dominio), hasta la utilización de algoritmos y técnicas adicionales desarrolladas de forma independiente. Por ejemplo, la incorporación de nuevas técnicas de exploración del espacio de soluciones, mejora de las obtenidas, que incluso podrán ser la aplicación de otras metaheurísticas, como las técnicas y mejoras presentadas en este documento.

Se entiende que no es casualidad que durante el desarrollo, evaluación funcional y de *performance* del sistema desarrollado, se haya notado que siempre se obtienen mejoras significativas al agregar conocimiento del dominio de interés a los operadores o técnicas utilizadas. Por lo cual, consideramos recomendable la hibridización del algoritmo genético con técnicas asociadas al problema que se está resolviendo y/o que agreguen conocimiento de interés a las mismas.

Finalmente, y apoyado en la evaluación de este trabajo, se considera importante destacar la relevancia de una buena calidad de datos de entrada del algoritmo para la obtención de buenos resultados. En este contexto, hay que tener en cuenta que esto incluye a todos los parámetros que se han presentado en la sección 3.6 y principalmente a la población inicial, la cual constituye la base de la evolución.

La aplicación de las técnicas de computación evolutiva, entre ellas los algoritmos evolutivos, han mostrado ser un conjunto de metaheurísticas efectivas y robustas para la resolución de problemas de optimización combinatoria. Considerando entre sus principales ventajas su alta aplicabilidad a múltiples áreas, son una alternativa para la resolución de problemas complejos de optimización relativos a los problemas de

planificación, ordenamiento y asignación de recursos, como el IDVSP, cuya resolución es uno de los objetivos principales de este proyecto.

9. Trabajos Futuros

9.1. Generador de Población Inicial

Tal como se ha visto en las secciones 3.6 y 7, correspondientes a la calibración y evaluación respectivamente, la generación de soluciones iniciales es un paso de gran relevancia e impacto en los procesos y resultados del algoritmo genético. En particular, se ha encontrado que, en ejecuciones cuya población inicial ha sido generada a través del método de entronques, y secuencial, se terminan obteniendo soluciones de mayor calidad que si se generara la población inicial utilizando el método aleatorio.

Por lo dicho anteriormente, se proponen dos posibles trabajos futuros en lo que respecta a la gesta de soluciones iniciales: En primer término se propone la creación de un generador de soluciones iniciales con mayor conocimiento acerca del problema, posiblemente un generador de soluciones iniciales que conozca las reglas laborales. Dicho generador podrá ejecutarse tanto independientemente o integrarse a los procesos ya existentes como procesos de pre o post procesamiento.

Por otro lado, otra posible mejora con respecto a las soluciones iniciales, sería agregar una funcionalidad que permita recibir las soluciones iniciales desde un archivo, por ejemplo, de esta manera se podría tomar cierta población que se sabe buena como base para una ejecución subsiguiente.

9.2. Ejecución Distribuida

Ya anticipado en el uso de la biblioteca MaLLBa y MPICH[MPICH], el diseño y modelo obtenido así como el motor de ejecución del sistema proveen la posibilidad de extensión para la ejecución en paralelo. Es decir, como un sistema distribuido en múltiples nodos admitiendo configuraciones tanto LAN como WAN. Las operaciones y métodos se proveen como componentes que extienden al motor (*Solver_LAN y Solver_WAN*) permitiendo así la orquestación de este tipo de ejecuciones.

Mediante la distribución de la carga en múltiples nodos, dicha funcionalidad podrá tener como objetivo, tanto disminuir los tiempos de respuesta, como incrementar los datos y procesamiento de los mismos manteniendo tiempos acordes, con la ventaja de poder explotar mejor la técnica. En particular, la posibilidad de ejecución de un algoritmo genético para poblaciones más numerosas o mayor cantidad de generaciones (evolución).

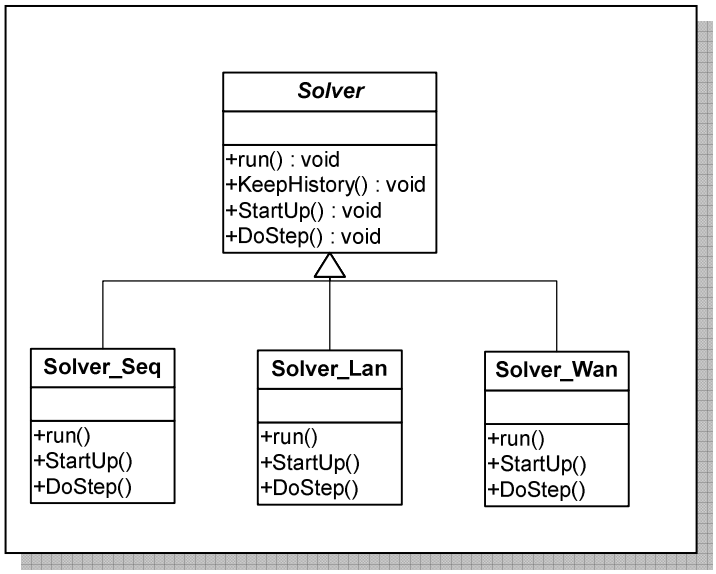

Fig. 9.2.1.1.Extensiones de la clase *Solver, Solver_Lan y Solver_Wan* para ejecución en paralelo.

Alguna de las pruebas y mejoras obtenidas de este tipo de ejecuciones se detallan en *Mallba: A library of skeleton for combinatorial optimisation* *[MALLBAOP]* donde se analizan las ventajas en tiempo de ejecución para una instancia del problema de asignación de recursos (*Resource Allocation Problem*[RAP]) donde se obtienen buenos resultados en cuanto a escalabilidad, con el incremento de unidades de procesamiento.

Stages-States	Sequential time (s) on fastest machine	Speed-up			
		2 procs. 700 MHz	4 procs. 700 MHz	4 procs. 700 MHz 4 procs. 500 MHz	4 procs. 700 MHz 9 procs. 500 MHz
1000-2000	457.79	1.97	3.92	4.12	6.01
1000-2500	714.87	1.98	3.94	4.30	6.02
1000-4000	1828.22	1.99	3.96	4.31	6.41
1000-5000	2854.04	1.99	3.97	4.24	6.42
1000-7000	5594.74	1.99	3.97	4.22	6.41
1000-10000	11422.60	1.98	3.97	4.18	6.38

Fig. 9.2.1.2 Resultados para el problema RAP utilizando un esqueleto MaLLBa en una red de 13PCs (4 AMD K6 700Mhz y 9 AMD K6 500Mhz) [RAP].

9.3. Extensión del Lenguaje de Reglas Laborales

Si bien la gramática definida para la especificación de las reglas laborales brinda una amplia gama de alternativas respecto a la posibilidad de extensión de componentes, considerando la complejidad del dominio y reglas, se plantea como trabajo futuro la incorporación de algún tipo de lenguaje declarativo al módulo de reglas laborales. Esto permitiría la declaración y accesibilidad a las distintas propiedades de las entidades, así como enriquecimiento del lenguaje a través de las posibles funcionalidades que el mismo provea. Un claro ejemplo de esta propuesta podría ser la incorporación o adaptación de OCL[OCL].

```
#El id de las categorias de un vehiculo es mayor que 2
context Vehicle
inv: self.category.id > 2

context Person
inv: self.fleet->select(v | v.category = alta)->size <= 3

#Todos los vehiculos con categoria <alta>
Vehicle.allInstances->select( v: Vehicle |
v.categoria=alta)

#Conjunto de Poligonos con un vertice en comun
Polygon.allInstances->iterate(
    p1 : Polygon ;
    cv : Set(Polygon) = Set{}
    |   if p1 <> p0 and
           (p1.vertices->intersection(p0.vertices))-
>notEmpty
        then cv->including(p1) else cv endif
    )
```

Fig. 9.3.1.Ejemplo de Object Constraint Language (OCL).

Fig. 9.3.1.Ejemplo de Object Constraint Language (OCL).

10. Referencias

[DESAUL] Public transit, Reporte Técnico G-2003-77, G. Desaulniers, M. Hickman (2003)

[BALL] A matching based heuristic for scheduling mass transit crews and vehicles. Transportation Science 17, 4–31 Ball, M., Bodin, L., Dial, R (1983)

[IAVCS] An integrated approach to vehicle and crew scheduling. Technical Report 9503/A, Economie Institute, Erasmus University Rotterdam, Rotterdam Freling, R., Boender, G., Paixao, J.M.P. (1995)

[IAECVSP] An integrated approach to the extra-urban crew and vehicle scheduling problem. In: Wilson, N.H.M. (ed.) Computer-Aided Transit Scheduling, pp. 103–128. Springer, Berlin Gaffi, A., Nonato, M.A (1999)

[NPHARD]Computers and Intractability—A Guide to the Theory of NP-Completeness. Freeman. Garey, M. R., D. S. Johnson (1979)

[METAH] International Journal of Metaheuristics (IJMHeur) ISSN (Online): 1755-2184- ISSN (Print): 1755-2176

[IUIO] Borrador Informe Proyecto de Grado" Fing Lab, interfaz de asistencia a la experimentación con algoritmia para el ordenamiento de vehículos y su tripulación en el transporte urbano colectivo" , Investigación Operativa, Instituto de Computación, Facultad de Ingeniería, UDELAR, Di Genio Mario (2010)

[MCPBO] Monte Carlo Simulation and Population-Based Optimization, Alain Cercueil Olivier François Laboratoire de Modelisation et Calcul, Institut IMAG, France, (2007)

[MIS] Metaheurísticas e Ingeniería del Software, Tesis Doctoral de José Chicano García dirigida por Enrique Alba, Universidad de Málaga (2007)

[MAPRRT] Metaheurísticas Avanzadas para Problemas Reales en Redes de Telecomunicaciones (2008)

[GLO] Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533-549 F. Glover (1986).

[TEVOL] Introduction to Genetic Algorithms, MIT Press, Cambridge, MA Mitchell, M. (1996)

[GOLD] *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Addison-Wesley, D. E. Goldberg (1989).

[HILLC] Alison Cawsey, HW University, (2010) http://www.macs.hw.ac.uk/~alison/ai3notes/subsubsection2_6_2_3_1.html

[BODIN83] Routing and Scheduling of Vehicles and Crews: The State of the Art. Computers and Operations Research, L. Bodin, B. Golden, A. Assad, and M. Ball. 10, 63–211. (1983)

[BERT87] On Some Matching Problems
Arising in Vehicle Scheduling Models. Networks, 17, 271–281. A.A. Bertossi, P. Carraresi, and G. Gallo (1987)

[LOBEL] Optimal Vehicle Schedulin in Public Transit. Andreas Lobel Dissertation (1997)

[MAIVCS] Models and Algorithms for Integration of Vehicle and Crew Scheduling. Journal of Scheduling 6, 63–85. Freling, R., D. Huisman, and A.P.M. Wagelmans (2003).

[MDIVCS] Multiple-Depot Integrated Vehicle and Crew Scheduling. Econometric Institute Report EI2003-02, Erasmus University, Rotterdam, The Netherlands. Huisman, D., R. Freling, and A.P.M. Wagelmans (2003).

[BDSUTS] Bus and Driver Scheduling in Urban Mass Transit Systems. Guy Desaulniers, (2002).

[PATHREL] Path Relinking Path Relinking For Multiple Objective Combinatorial Optimization. TSP Case Study, Andrzej Jaszkiewicz, http://www.iasi.cnr.it/ewgt/16conference/ID155.pdf

[PAIX] Multiobjective Metaheuristics for the Bus Driver Scheduling Problem, Helena R. Lourenço, José P . Paixão, Rita Portugal (2001)

[VMWARE] VMware software de virtualización - www.vmware.com

[DELL] Dell Hardware - www.dell.com

[INTEL] Intel Corporation, the World leader in Silicon Innovation - www.intel.com

[GABDSP] Genetic algorithms for the bus driver scheduling problem: a case study, Universidade do Porto/INEGI, Porto Portugal, Dias, De Sousa, Cunha (2002)

[HAO08] Simultaneous Vehicle and Crew Scheduling for Extra Urban Transports, Jin-Kao Hao, Benoit Lauren (2008)

[SETPA] Set Covering, Packing and Partitioning Problems, Karla Hoffman (GMU), Manfred Padberg (NYU) (2010) http://iris.gmu.edu/~khoffman/papers/set_covering.html

[XML] Extensible Markup Language http://www.w3schools.com/xml/default.asp

[DLL] Dynamic Link Library http://www.cygwin.com/cygwin-ug-net/dll.html

[DOO] Object Oriented Modeling and Design - Prentice Hall - ISBN 0–13–629841–9, James Rumbaugh

[MALLBA2] MaLLBa - http://www.lsi.upc.edu/~mallba/public/

[MALLVRP] Solving Complex Problems with advanced Techniques, Enrique Alba http://www.ercim.eu/publication/Ercim_News/enw56/alba.html

[MPICH] MPICH, Implementation of MPI - http://www.mcs.anl.gov/research/projects/mpi/mpich1/

[MPI] Message Passing Interface - http://www.mcs.anl.gov/research/projects/mpi/

[COPSA] Cooperativa de Ómnibus de Pando Sociedad Anónima. www.copsa.com.uy

[MALLBAOP] MaLLBa: A library of skeletons for combinatorial optimization, Enrique Alba et al www.lcc.uma.es/~eat/pdf/europar2.pdf

[RAP] *Enumerative Approaches to Combinatorial Optimization, Part II*. Annals of Operations Research. Volume 11, 1-4, T. Ibaraki (1988).

[OCL] Object Constraint Language - http://www.omg.org/technology/documents/formal/ocl.ht

A. Gestión y Desarrollo del Proyecto

A.1. Introducción

El propósito de esta sección es la presentación y descripción de los requerimientos planteados así como los resultados esperados en cuanto al sistema a desarrollar para la problemática planteada. También se incluyen los aspectos principales de la metodología y desarrollo del proyecto.

A.2. Objetivos y Requerimientos

A.2.1. Objetivos

Para este proyecto se plantean principalmente dos desarrollos concretos:

- Implementación de una metaheurística dada para resolver el problema de optimización combinatoria de planificación de libros de servicio.
- Diseño e implementación de un módulo de especificación de reglas laborales.

Se trabajará con casos de estudio de diferentes características y tamaños. El proyecto se enmarca en un proyecto de largo alcance, referente a la planificación operacional de sistemas de transporte público urbano colectivo. Si bien cada empresa le impone al problema particularidades propias, podemos definir resumidamente:

Asignación de Flota: Consiste en asignar los vehículos disponibles a la realización de los viajes de cada línea, programados a lo largo del día, teniendo en cuenta características como existencia de más de un depósito y múltiples tipos de vehículos, minimizando el total de vehículos necesarios y los tiempos no productivos de los mismos (viajes expresos, esperas, etc.).

Asignación de Personal: Consiste en asignar el personal a la operación de los vehículos asignados a la realización de viajes en la asignación de flota, respetando las reglas laborales, minimizando aspectos como las horas extras, cambios de vehículos y cambios de líneas.

Los resultados esperados cubren:

- Desarrollo de una metaheurística que procure resolver el problema planteado.
- Diseño e implementación de la solución al problema planteado.
- Generación de un módulo de descripción de reglas laborales para su uso en la algoritmia de optimización.

- Experimentación y relevamiento del sistema desarrollado.

- Integración a la interfaz de visualización del Departamento de Investigación Operativa de la Facultad de Ingeniería.

A.2.2. Requerimientos

Se plantea el estudio e implementación de un algoritmo para la planificación óptima de libros de servicio para sistemas de transporte público colectivo. El problema consiste en: dados un conjunto de viajes a realizar, una flota de ómnibus disponibles y un conjunto de reglas laborales, determinar la secuencia de viajes que debe realizar cada vehículo para cumplir los viajes, respetando las reglas laborales de sus tripulantes con el objetivo de minimizar los costos definidos.

Se desea que el sistema sea capaz de distribuir la flota de ómnibus, al conjunto de viajes que han sido asignados para un determinado día de trabajo, de forma de reducir los costos de la empresa. En otras palabras, se deberán planificar los servicios tratando de reducir principalmente (entre otros posibles intereses) la suma ponderada de:

- La cantidad de kilómetros expresos

- La cantidad de horas asignadas

- La cantidad de jornales de N horas

Restringido a un conjunto de reglas y normas laborales, donde además, se proveerá un módulo para la especificación y parametrización de dichas reglas que permita realizar las modificaciones y/o correcciones correspondientes de modo de poder ejecutar y obtener nuevas soluciones con modificaciones y/o variantes propias de la realidad en cuestión.

A.2.2.1. Interfaz de Usuario

El sistema a construir le proporcionará al usuario como resultado de su ejecución los libros de servicio, es decir, la asignación correspondiente al problema de ordenamiento de la flota de modo de cubrir los viajes indicados, así como los turnos generados para cumplir con los requerimientos de normativas laborales definidos para la tripulación.

Se plantea un formato de salida análogo al utilizado por la compañía, el cual consta de tablas de asignación de viajes y horarios, por vehículo, turnos y servicios.

La configuración de parámetros y datos de entrada será a través de archivos de configuración con formato definido.

También se anticipa facilitar la compatibilidad con la interfaz de usuario actualmente siendo desarrollada por el Departamento de Investigación Operativa [IO].

A.2.2.2. Interfaz de Comunicación

Se define y provee una interfaz (*IWorkerController*) de comunicación para el módulo de reglas laborales de modo que pueda ser utilizado independientemente.

A.2.2.3. Restricciones de Memoria

Dados los tipos de datos manipulados, la memoria de almacenamiento requerida no debería representar un obstáculo para cualquier equipo estándar, por lo que no existen requerimientos particulares de esa índole.

Aún así, dada la naturaleza del problema y posibles estrategias planteadas para la resolución del mismo, el uso de recursos de procesamiento, incluyendo memoria (RAM), deberá ser considerado como crítico buscando hacer un uso eficaz y eficiente de los mismos.

A.2.2.4. Requerimientos de Adecuación al Entorno

No se identifican requerimientos adicionales de este tipo que escapen a las correspondientes guías de instalación e implantación del sistema en cuestión. En particular se propone la utilización de una maquina virtual generada y soportada por VMWare[VMWARE] de modo de asegurar la portabilidad y adecuación a distintos entornos físicos en caso de ser necesarios.

A.2.2.5. Funciones del Producto

En esta sección se resumen las funciones más importantes que el software debe realizar. Este será organizado en dos subsistemas o módulos principales como se detallan a continuación, donde existirá una relación de visibilidad unidireccional desde el módulo de resolución hacia el módulo de reglas laborales de modo de cumplir con los requerimientos planteados en cuanto a la obtención de soluciones restringidas a reglas laborales.

A.2.2.5.1. Módulo de Resolución del Algoritmo

Este módulo deberá proveer la implementación de la solución/algoritmo planteado para la generación de los libros de servicio considerando la realidad planteada y restringida a las reglas laborales definidas las cuales serán interpretadas a través de la interacción con el Módulo de Reglas Laborales.

Automatización de Generación de Soluciones Iniciales

Este requerimiento surge a raíz de los procedimientos y técnicas actualmente utilizados por el grupo de tránsito de COPSA para la generación manual de libros de servicio. Dada la utilidad que demuestra empíricamente el uso de dichas técnicas con el

fin de obtener libros de servicio de bajo costo, se propone la inclusión de dicha funcionalidad donde se automatiza el proceso de modo de obtener tales soluciones.

Integración al Módulo de Interfaz Gráfica

Se solicita la integración de las soluciones generadas con un sistema existente de despliegue gráfico de soluciones que permite visualizar las asignaciones mediante la utilización de Diagramas de Gantt[GANTT].

A.2.2.5.2. Módulo de Reglas Laborales

Este módulo deberá proveer funcionalidades en cuanto a la definición, gestión y validación de reglas laborales.

Definición de Reglas Laborales

El módulo deberá permitir funcionalidades de definición y extensión de reglas laborales ya sea mediante la definición o interpretación de algún lenguaje que modele dicha realidad.

Gestión de Reglas Laborales

Así como la definición de las mismas, el módulo deberá estar diseñado de manera de permitir fácilmente la modificación de valores y/o redefinición de reglas laborales.

Validación de Reglas Laborales

Dada una solución, cuya representación será conocida, el módulo deberá ser capaz de determinar la factibilidad y/o cumplimiento del conjunto de reglas consideradas.

A.2.2.6. Características de los Usuarios

Si bien el producto a desarrollar es para el grupo de Investigación Operativa[IO] de la Facultad de Ingeniería, se procurará que el mismo pueda ser configurado y ejecutado por usuarios no necesariamente expertos en informática, sino que tengan dominio de la realidad planteada y posean conocimiento básicos para la edición de los archivos de configuración.

A.2.2.7. Restricciones de Diseño

En esta sección se describen los elementos que limitan las opciones del desarrollo. Estas restricciones representan decisiones de diseño requeridas o que han sido tomadas luego de los correspondientes relevamientos.

A.2.2.8. Lenguaje de Programación

A pesar de no existir restricciones en cuanto al lenguaje a utilizar, motivado principalmente por el requerimiento de eficiencia planteado, se opta por la utilización de *C++* como lenguaje de programación ya que no solo resulta ser conocido por el grupo de desarrollo sino que entre muchas de sus ventajas se encuentra la posibilidad de optimización de estructuras y algoritmos auxiliares a utilizar considerando principalmente el desarrollo de la estrategia o algoritmo.

A.2.2.9. Plataforma

Dadas las ventajas implícitas que ofrece una plataforma libre y abierta, se selecciona Linux como sistema operativo, particularmente la distribución *Ubuntu*[UBUNTU] 8.04, dada la estabilidad que ha demostrado dicha versión con respecto a las funcionalidades a utilizar.

A.2.2.10. Herramientas de Desarrollo

Considerados los requerimientos previamente mencionados y teniendo como preferencia el uso de herramientas libres, se selecciona *Eclipse IDE for C/C++* como entorno de desarrollo, en su última versión estable, *Eclipse-cpp-galileo-SR1*[ECLIPSE].

Para la gestión de configuración se utilizará un repositorio *SVN Subversion*[SUBVER] con las correspondientes extensiones para su integración al entorno: *Subclipse*[SUBCLI], *TortoiseSVN*[TORTOISE] y *RapidSVN* [RAPID].

A.2.2.11. Biblioteca MaLLBa

Considerando el problema a resolver y posibles estrategias a desarrollar, se propone la utilización de una biblioteca reconocida en el ámbito académico como ser MaLLBa[MaLLBa]. Entre los motivos que impulsan el uso de la misma se encuentran: experiencias satisfactorias (ajenas y propias) en cuanto a su uso y resultados obtenidos que han probado su eficacia y eficiencia. Así como también el hecho de ser una biblioteca libre y abierta con un diseño simple y posibilidades de modificación, adaptación y extensión de cualquiera de sus componentes. Además MaLLBa provee compatibilidad con MPICH[MPICH], una implementación de MPI[MPI] que permite la posibilidad de computación paralela. Dada la importancia de esta biblioteca por detalles sobre la misma referirse al anexo D.

A.2.2.12. Requerimientos Suplementarios

Se pretende que la solución tenga un alto grado de parametrización en cuanto a definición de valores y constantes, permitiendo su mantenibilidad y posible extensión. El objetivo del módulo de resolución no solo se centrará en minimizar tiempos de respuesta sino también en la eficacia y eficiencia de las estrategias desarrolladas desde el punto de vista del uso de recursos.

A.2.2.13. Documentación

Se proveerá toda la documentación correspondiente al sistema desarrollado incluyendo aspectos como ser: descripción de la realidad y factores considerados, decisiones y estrategias tomadas, diseño y modelo construido, documentación técnica sobre el desarrollo e implementación del sistema e informe y reportes sobre los distintos casos de estudio y resultados obtenidos.

A.2.2.14. Guía de Instalación y Configuración

Se proveen guías que cubren los aspectos relevantes acerca de la configuración para la especificación de datos de entrada y ejecución de cualquiera de los módulos que componen al sistema desarrollado.

También se incluirá una guía de instalación del entorno y puesta a punto del sistema así como guías de configuración y ayuda embebida en el mismo procurando ser una referencia de rápido acceso.

A.3. Metodología de Desarrollo

Tratándose de un problema muy restringido, es de esperar que pequeñas variaciones, tanto en la realidad considerada como en lo parámetros de configuración, pueden afectar considerablemente los resultados obtenidos a través de las distintas técnicas utilizadas. Esto, sumado a la dificultad implícita que tiene la evaluación y comparación de soluciones para distintos escenarios son factores a tener en cuenta para la obtención de buenos resultados durante la evolución y desarrollo de la solución.

El desarrollo del proyecto se lleva a cabo por etapas semejantes a un modelo en cascada, donde en cada una de ellas se procuran resolver problemáticas particulares considerando la anterior como base. Dicha modalidad surge a raíz de la necesidad de un modelo iterativo e incremental que permita de alguna manera asegurar que la solución desarrollada resuelve lo considerado hasta el momento (dentro de los parámetros de validez definidos) de manera de poder concentrarse en la mejora e inclusión de las nuevas funcionalidades a incluir en la etapa actual, pudiendo así aislar y mitigar los riesgos del impacto de errores generados por las nuevas incorporaciones.

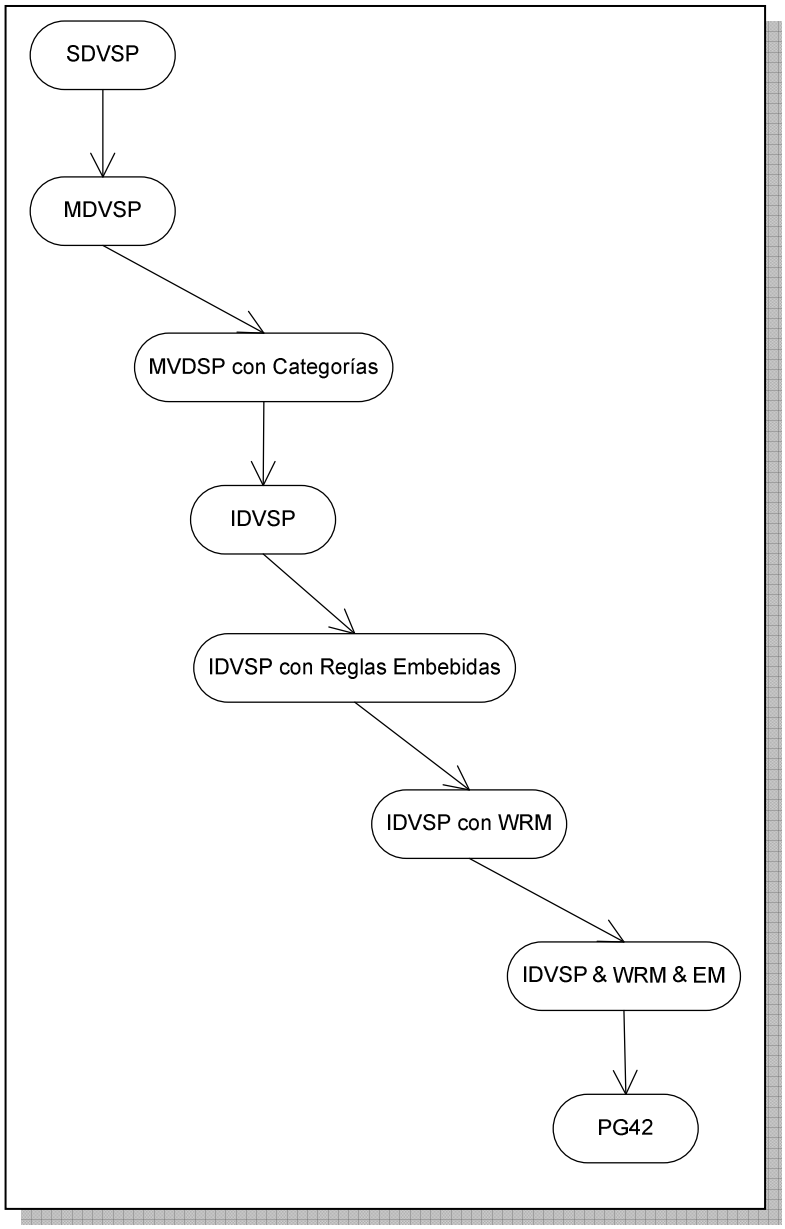

Fig. A.3. Etapas y problemas considerados durante la evolución del proyecto.

A.3.1. SDVSP

Considerada como etapa inicial de la implementación del sistema, el desarrollo del SDVSP[SDVSP] enmarcado como proyecto del curso de Algoritmos Evolutivos[ALGEVOL] propone el desarrollo de una solución para el problema aplicando dicha metaheurística. Esta etapa permitió ganar experiencia en el modelado e implementación de la técnica así como relevamiento de bibliotecas existentes para la aplicación de estas, donde particularmente se opta por el uso de MaLLBa.

A.3.2. MDVSP

El MDVSP[MDVSP93] representa un problema más general del SDVSP ya que este último puede considerarse como un caso particular del primero. Incluso se podría plantear la posibilidad de resolución del MDVSP como múltiples instancias del SDVSP que podrían resolverse independientemente para luego integrarlas, procurando hacerlo de manera eficiente en búsqueda de soluciones al problema total. Pese a que esa modalidad no es la deseada, la evolución a esta etapa es casi implícita si se pretende llegar a modelar la realidad planteada. Se propone como objetivo, la obtención de soluciones para el problema de múltiples depósitos, así como la verificación de las técnicas seleccionadas para la resolución de este problema.

En esta experiencia, aparte de la verificación y validación del modelo y técnicas, surgen importantes conclusiones en cuanto a la influencia e impacto de las soluciones iniciales (*initial population*) sobre los resultados obtenidos. Se observa que podrán obtenerse mejores soluciones, en generaciones más tempranas mediante el uso de técnicas de inicialización ajustadas al escenario particular del problema.

También se concluye que, pese a que el costo de evaluación y procesamiento de mantener y trabajar solamente con individuos factibles puede generar dudas en cuanto a la eficiencia, los resultados obtenidos indican que, mediante la aplicación de técnicas que permitan evaluar esta propiedad de manera eficiente, podemos obtener enormes beneficios en cuanto a la construcción y búsqueda de soluciones óptimas. Esto se debe a que dadas las múltiples restricciones en cuanto al formato de la misma, las técnicas clásicas de evolución de los individuos podrían generar pérdidas y costos mayores, e incluso derivar en una población enteramente no factible.

A.3.3. MDVSP con Categorías

La inclusión de categorías al problema, viene a la necesidad y hecho de poder considerar a la flota disponible como heterogénea en cuanto a tipos y preferencias de asignación a los viajes a cubrir. Este tipo de restricciones puede estar ligado tanto a normas de tránsito particulares que establezcan restricciones, a los tipos de vehículos que pueden circular por determinadas zonas, o preferencias u objetivos comerciales de la propia empresa como ser, disponer de mejores unidades para viajes donde existe mayor competencia.

En esta etapa se plantea la inclusión de tales conceptos de manera de poder establecer restricciones sobre las unidades a asignar a los viajes. Teniendo en cuenta que dependiendo de la cantidad de unidades disponibles y dimensión del problema, la restricción puede tener un impacto determinante en la obtención de soluciones factibles. Para esto, se establece un mecanismo de prioridades en cuanto a preferencia, que permite de alguna manera abrir el espectro de opciones de asignación durante la construcción de soluciones, tanto iniciales como durante la propia evolución.

A.3.4. IDVSP

Una vez validada la etapa anterior, se retoma uno de los requerimientos principales, que consta de la inclusión de conceptos relativos a las normas laborales en la problemática agregándose así la asignación de tripulación.

Siendo la primera aproximación al problema integrado, se incorporan conceptos básicos de la generación de los libros de servicios, como ser las duraciones estipuladas de los turnos y su cantidad máxima (por vehículo). Es aquí, donde surgen dos funcionalidades en cuanto a las reglas laborales. Por un lado, la necesidad de integrar los conceptos al algoritmo de manera que la búsqueda y evolución de los individuos genere soluciones que respeten de manera restrictiva las normas establecidas, así como también optimizar las soluciones generadas. Por otro lado, la necesidad de contar con algún mecanismo que permita la especificación e inclusión de dichas reglas al algoritmo. De lo anterior, derivan las etapas subsiguientes, en donde particularmente como se establece en la sección 4.3.1, se opta por la inclusión de algunos conceptos claves en las técnicas de generación de turnos, de manera de sacar mayor provecho al conocimiento de esa información para poder así realizar asignaciones de tripulación de manera más eficiente minimizando los costos de ejecución. Una vez verificado y validado el funcionamiento y obtención de soluciones restringidas a las reglas embebidas, se procede a incluir el módulo de reglas laborales (WRM), el cual contiene en sí las funcionalidades de gestión y evaluación de las reglas laborales definidas por el usuario.

Esta etapa incluyendo sus derivadas representan las más exigentes en cuanto a desarrollo y verificación, por lo que se establece el siguiente plan de manera de fijar actividades y camino crítico.

A.3.4.1. Plan de Desarrollo del Sistema Integrado

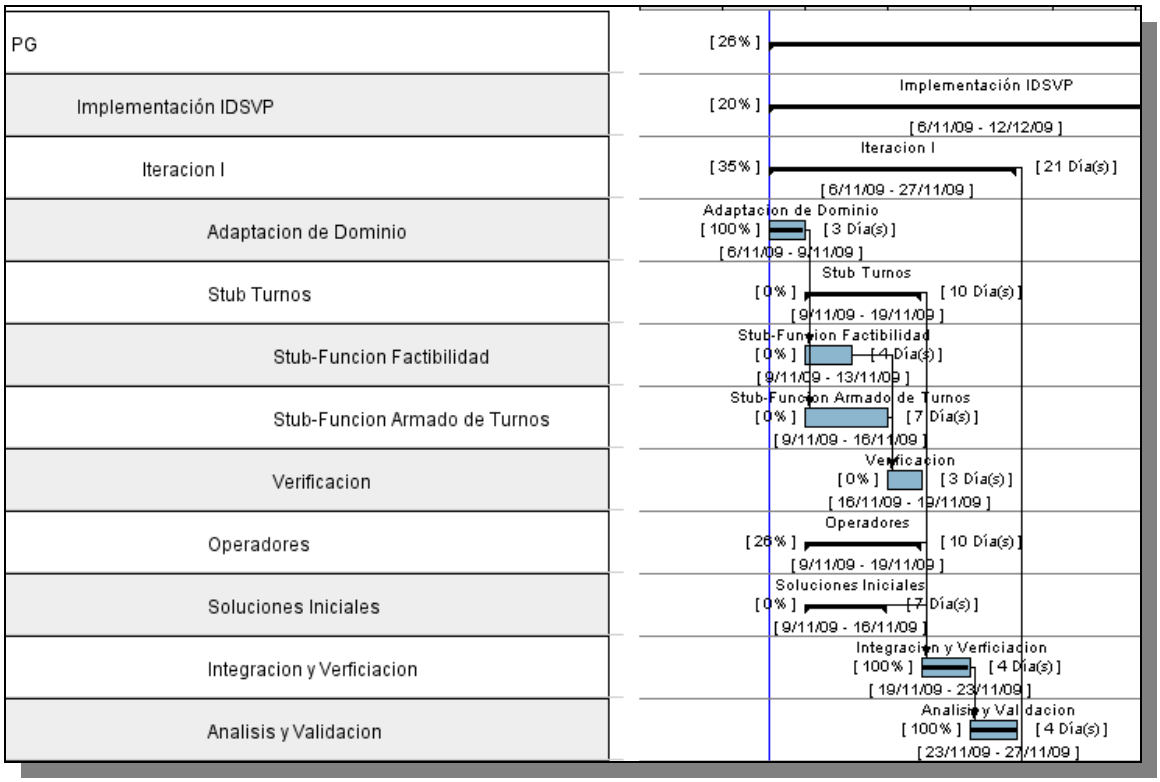

Fig. A.3.4.1.a Diagrama de Gantt para el desarrollo del IDSVP & WRM (*Plan_1.png*).

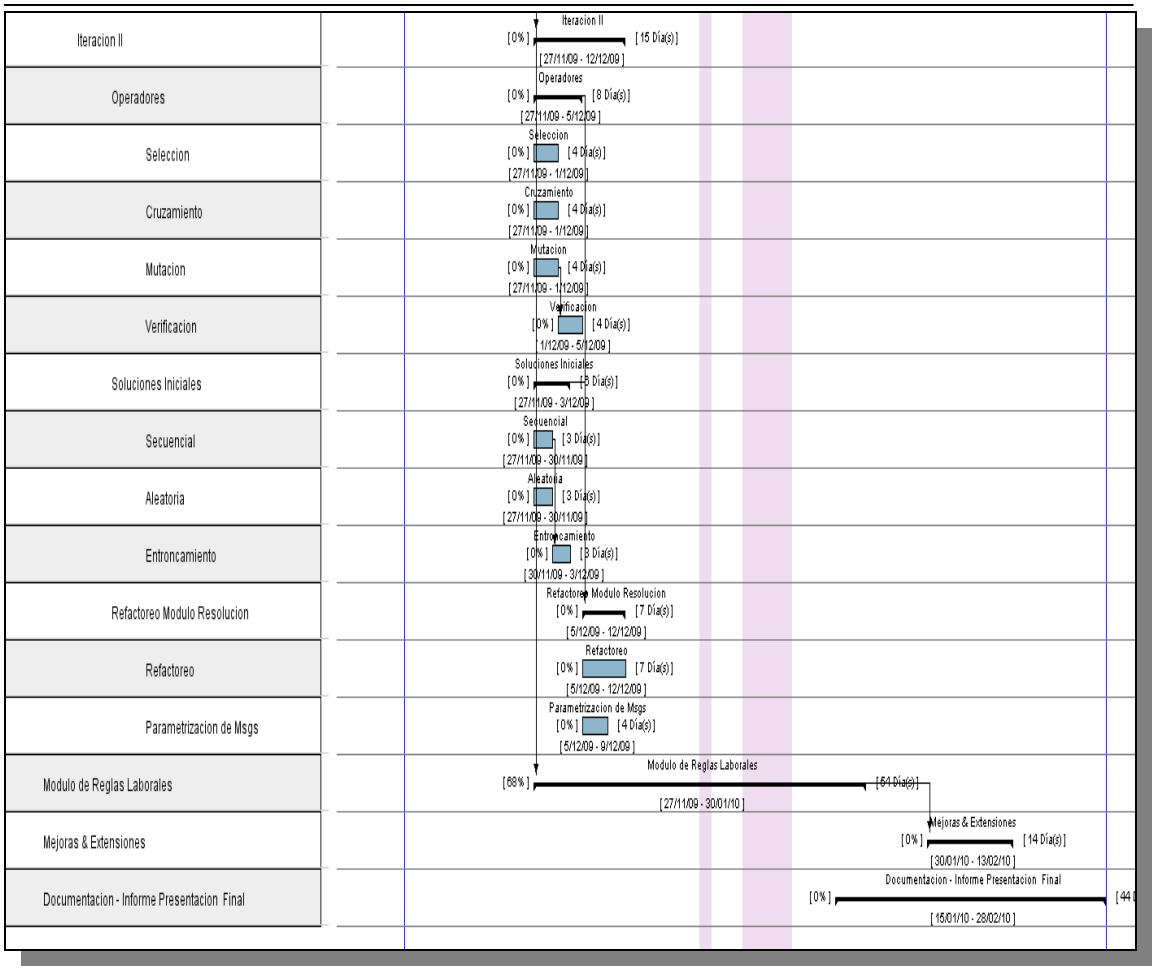

Fig. A.3.4.1.b Diagrama de Gantt para el desarrollo del IDSVP & WRM (*Plan_2.png*).

A.3.4.2. Diagrama PERT

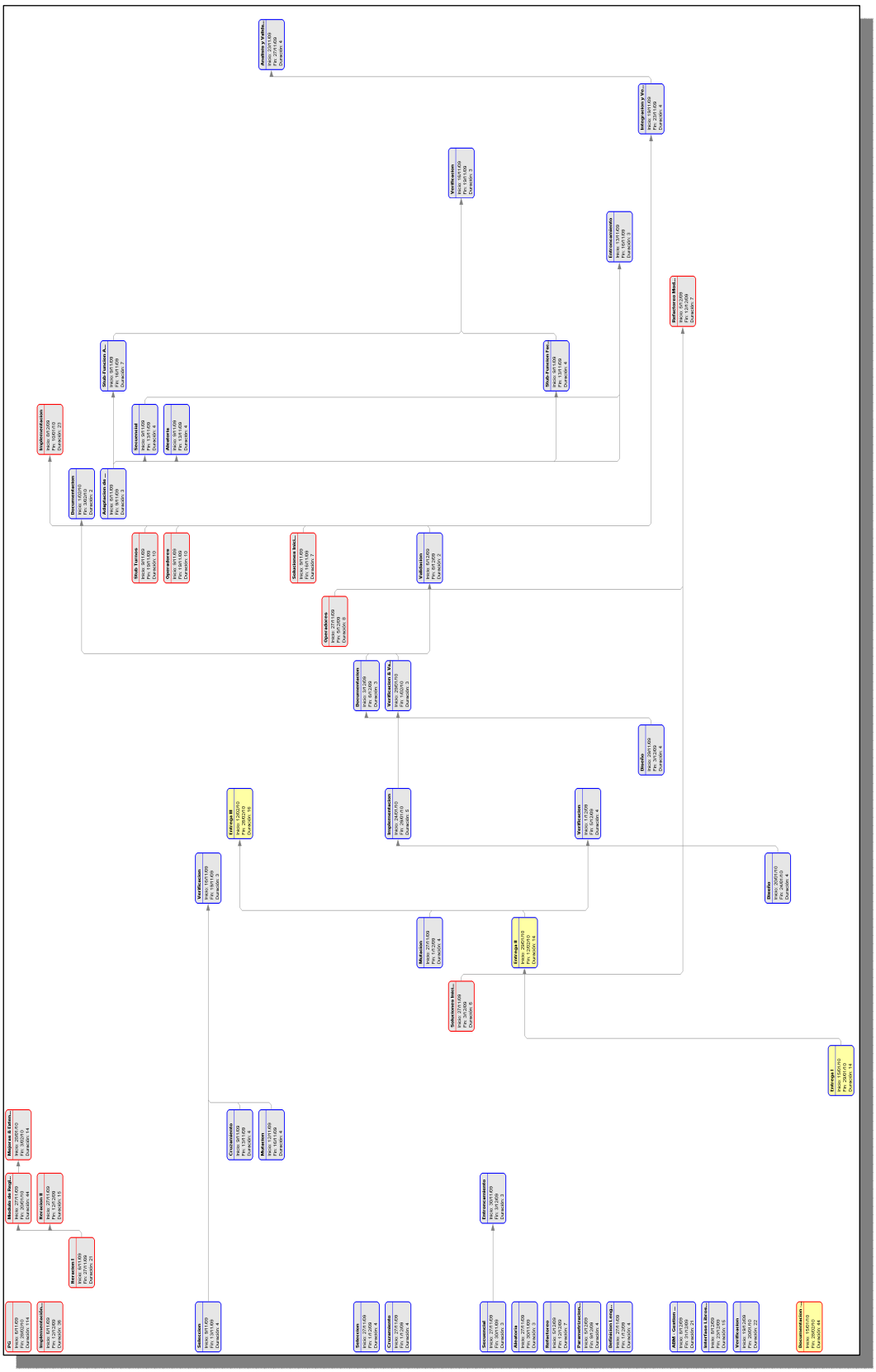

Fig. A.3.4.1. Diagrama PERT para el desarrollo del IDSVP & WRM.

A.3.5. IDVSP, WRM y EM

La última etapa de desarrollo del proyecto consta de la integración de todos los módulos, teniendo como agregado el Módulo de Extensiones (EM), el cual como su nombre lo indica permite el agregado de extensiones que en sí representarán *pre* y *post* procesamientos de procesos claves y definidos del sistema. Particularmente, como se detalla en la sección 5, para el sistema final se incluye un post procesamiento de la generación de turnos que busca optimizar la asignación generada por el WRM, obteniendo mejores resultados en las evaluaciones realizadas cuyo detalle se presenta en la sección 7.

A.3.6. PG42

PG42 representa la etapa o estado final del proyecto como producto, siendo éste el sistema integrado y operacional en su totalidad. En esta etapa no se realizan más que ajustes de calibración y parámetros de manera de poder realizar la evaluación final especificando y determinando los escenarios deseados.

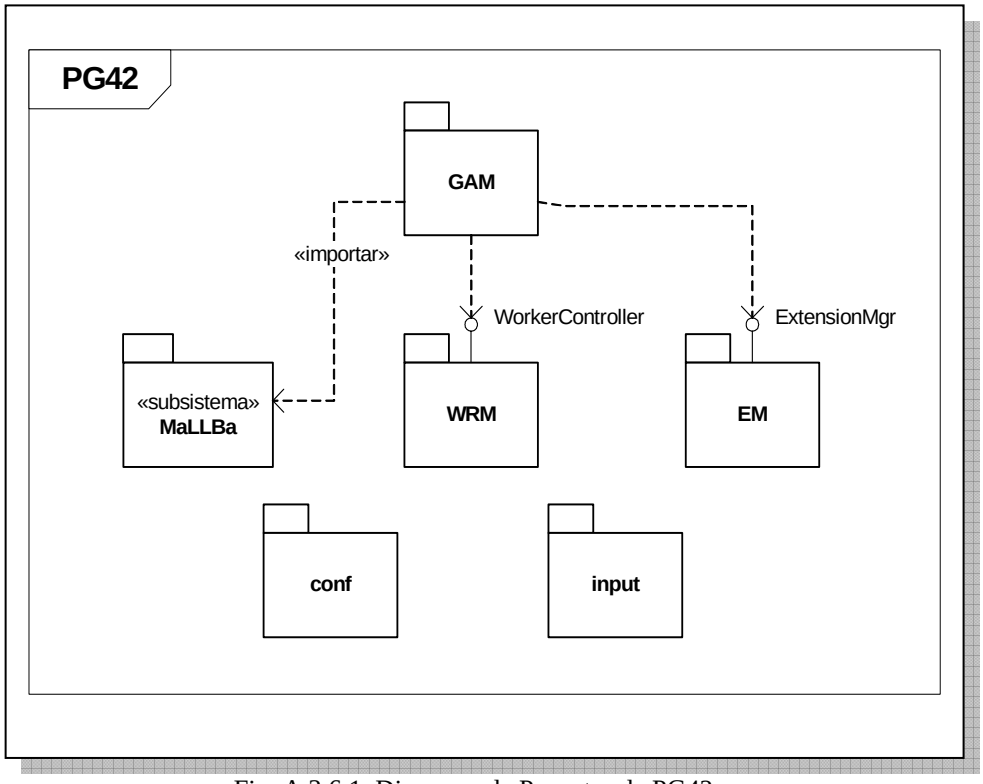

Fig. A.3.6.1. Diagrama de Paquetes de PG42.

El diseño obtenido organiza los distintos módulos por paquetes, siendo el módulo del algoritmo (GAM) quien provee el ejecutable (PG42) del sistema. Los paquetes *conf* e *input* son los contenedores de los archivos de especificación de la configuración del sistema y datos de entrada respectivamente.

A.4. Referencias

[IO] Grupo de Investigación Operativa - http://www.fing.edu.uy/inco/grupos/invop/
[VMWARE] Virtual Machine - http://www.vmware.com
[GANTT] Diagrama de Gantt - http://www.colostate.edu/Services/ACNS/swmanuals/sasdoc/sashtml/orpm/chap4/index.htm
[UBUNTU] Ubuntu - http://www.ubuntu.com/
[ECLIPSE] Eclipse IDE - http://www.eclipse.org
[SUBVER] SVN SubVersion - http://subversion.tigris.org/
[SUBCLI] Subclipse - http://subclipse.tigris.org/
[TORTOISE] Tortoise SVN - http://tortoisesvn.tigris.org/
[RAPID] Rapid SVN - http://rapidsvn.tigris.org/
[MaLLBa] MaLLBa - http://www.lsi.upc.edu/~mallba/public/
[MPICH] MPICH - http://www.mcs.anl.gov/research/projects/mpi/mpich1/
[MPI] Message Passing Interface - http://www.mcs.anl.gov/research/projects/mpi/
[SDVSP] Single Depot Vehicle Scheduling Problem – Jaques A.Ferland 1986, ISBN 978-3-540-17083-9
[ALGEVOL] Curso de Algoritmos Evolutivos, FING, UdelaR (2010) http://www.fing.edu.uy/inco/cursos/geneticos/ae.html
[MDVSP93] Multiple Depot Vehicle Scheduling Problem – Dell'Amico, Fischetti, Toth, (1993) -http://www.jstor.org/pss/2661525

B. Módulo de Algoritmo Genético (GAM)

B.1. Introducción

El módulo de resolución del algoritmo genético (GAM) modela la realidad planteada e implementa un algoritmo genético de modo de buscar soluciones óptimas.

Este documento incluye los fundamentos, detalles y aspectos técnicos de las estrategias, diseño y funcionalidades implementadas en este módulo.

B.2. Objetivos y Requerimientos

B.2.1. Objetivos

Su objetivo principalmente es la búsqueda y generación de soluciones factibles de la mejor calidad posible según los costos definidos, condicionas a las restricciones de ordenamiento propias del problema, así como la integración de las normativas laborales mediante la inclusión del módulo de reglas laborales (WRM).

B.2.2. Requerimientos

Se requiere un sistema que provea la implementación de una metaheurística aplicada al problema de ordenamiento de flota y asignación de tripulación que genere los libros de servicio cumpliendo con normativas laborales definidas en un módulo independiente.

Entre los requerimientos principales se encuentran:

- Obtención de soluciones factibles
- Parametrización de coeficientes, constantes y valores de interés.
- Optimización de soluciones de acuerdo a ponderación variable de factores de interés.
- Integración de reglas laborales con bajo acoplamiento.
- Eficiencia y minimización de tiempos de ejecución.
- Integración a la interfaz gráfica provista por el Dpto. de Investigación Operativa.

B.3. Análisis y Diseño

B.3.1. Consideraciones Generales

El resultado del diseño y desarrollo del GAM es el producto de una evolución iterativa e incremental de soluciones desarrolladas a lo largo del proyecto para problemas semejantes incrementando su complejidad partiendo desde el SDVSP, MDVSP, MDVSP con categorías, IDVSP para llegar al diseño final de dicho módulo.

Tomando como base el esqueleto de algoritmo genético provisto por la biblioteca MaLLBa[MaLLBa], se procedió a especificar y extender conceptos para modelar la realidad descrita. Un esqueleto es una pauta algorítmica que implementa un método genérico de resolución, y que debe ser debidamente completado para instanciar y resolver un problema concreto, los detalles de estos se describen en el anexo D.

El diseño e implementación de la solución incluyendo dominio, componentes del algoritmo y motor de ejecución, se basa en el paradigma de programación orientada a objetos[OOP 07] [OOPA 03] no solo por sus ya conocidas ventajas en cuanto a modularización y extensibilidad, sino también por su conocimiento del grupo de desarrollo y uso en la biblioteca utilizada MaLLBa.

B.3.2. Organización

Los componentes de este módulo pueden ser categorizados de las siguientes maneras. Por un lado, clases de dominio, que corresponden al modelo de las entidades relevantes en la realidad planteada, y en segundo lugar, las relativas al algoritmo genético y motor de ejecución de éste.

B.3.2.1. Dominio

Componen las clases correspondientes al modelo de la solución y entidades pertinentes al modelado del problema. Se pretende y apunta a obtener un modelo comprensible para su mantenimiento, extensión y adaptación a distintas realidades donde se modelan conceptos de alto nivel que permiten su ajuste y extensión a distintos escenarios.

B.3.2.1.1. Modelo de Dominio

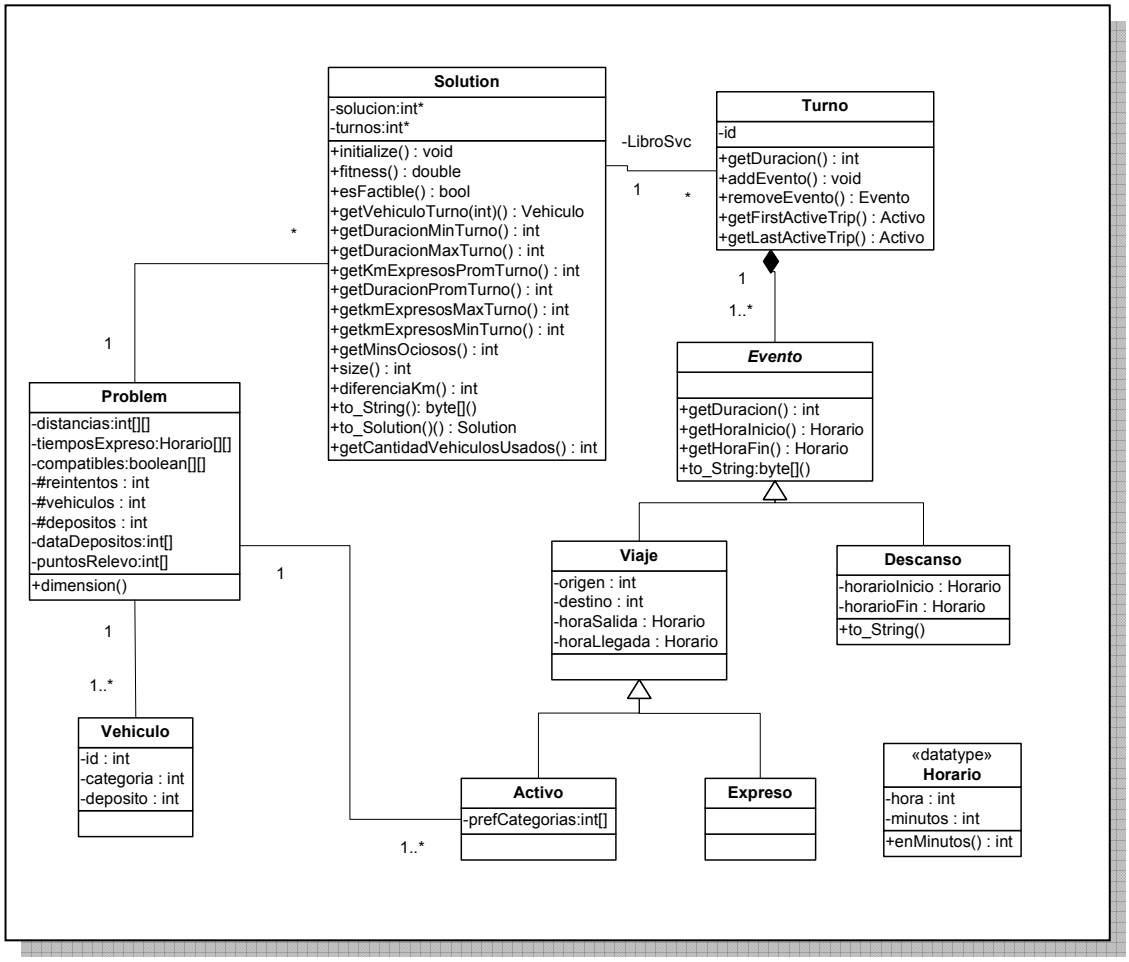

Fig. B.3.2.1. Diagrama de Modelo de Dominio

B.3.2.1.2. Clase *Solution*

Corresponde a uno de los conceptos y clases más importantes ya que define la representación y métodos de una solución a un problema de optimización para ser interpretada por el motor de ejecución del algoritmo. En ella se definen e implementan algunas de las funciones más relevantes, como son la de *fitness* y la de inicialización de soluciones y por ello es requerida por el motor de ejecución del algoritmo.

La solución puede dividirse en dos partes principales. Una de ellas, y quizás la más elemental, es la que contiene los vectores de asignación. Esta se corresponde con la codificación de la solución (cromosoma) necesaria para la ejecución del algoritmo genético.

- *int* solucion* – Vector que indica el vehículo asignado a cada viaje
- *int* turnos* – Vector que indica el turno asignado a cada viaje

Ambos vectores con tamaño igual a la cantidad de viajes a ordenar (*CANT_VIAJES)*, establecen las asignaciones de vehículos y turnos a viajes respectivamente. Este tipo de formato, que se entiende como simple dentro de los analizados, permite indizar tanto vehículo y turnos por viaje permitiendo un acceso rápido a dicha información *O(1)* y facilitando la lógica y operativa de los operadores genéticos. Dado que se incluye la completitud o cobertura total de los viajes en la factibilidad de las soluciones este formato permite fácilmente establecer particiones del conjunto de viajes y procesar las mismas asegurando dichas restricciones.

En segundo lugar se incluyó lo que se denomina el Libro de Servicios de la solución. Cada libro es generado a partir de los vectores previamente mencionados a través de la funcionalidad de generación de turnos brindada por el módulo de reglas (WRM). Este es compuesto por un conjunto de servicios que en sí, son representados a través de una lista de turnos con los detalles sobre los eventos a realizar, incluyendo viajes activos, viajes expresos, y descansos.

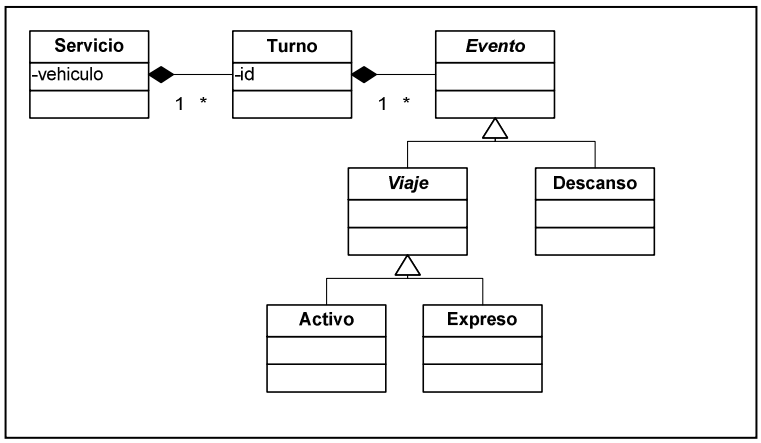

Fig. B.3.2.1.2. Composición de un servicio

B.3.2.1.3. Clase *Problem*

Esta clase es otra de las clases requeridas y básicas, ya que define la interfaz mínima necesaria para representar un problema de optimización, más concretamente, el problema a resolver. En ella se definen todas las estructuras y variables que son necesarias tener en cuenta para poder obtener una solución óptima, entre ellos datos de entrada, datos calculados a partir de estos y parámetros del problema particulares a la instancia a ejecutar. Dado que se trata más que nada de un descriptor del problema dicha clase es instanciada una única vez y referenciada por los individuos de la población de modo de minimizar réplicas innecesarias.

Entre sus componentes principales se encuentran:

- *int[][] distancias* – Corresponde a la tabla de distancias entre nodos.

- *Horario[][] tiemposExpreso* – Contiene la duración estimada de realizar un viaje expreso (*deadhead trips*) entre los nodos *[i][j]*

- *boolean [][] compatibles* – Matriz *boolean* que establece la compatibilidad entre dos viajes *[v][b]*, la misma es calculada para cada instancia de acuerdo a las restricciones de ordenamiento de flota.

- *int[] dataDepositos* – Vector que indica la cantidad de unidades disponibles por depósito.

- *int[] puntosRelevo* – Vector que contiene un mapeo de los nodos que indica si son puntos de relevo.

- *Vehiculo[][] vehiculos* – Contiene los datos de todos los vehículos disponibles en la flota.

- *Activos [][] viajes* – Contiene todos los viajes activos a cubrir (*active trips).*

B.3.2.1.4. Clase *Turno*

La clase turno se corresponde con el jornal laboral de un tripulante o tripulación donde se llevarán a cabo determinadas actividades. La misma contiene datos pertinentes a la identificación y manejo de una colección de eventos que especifican el tipo de actividades a desarrollar. Cabe aclarar que cada turno pertenece a un único libro de servicio de una única solución o individuo de la población. Los servicios que componen al libro se obtienen de la recopilación de los turnos realizados por un mismo vehículo.

B.3.2.1.5. Clase *Evento*

La clase abstracta *Evento* permite justamente la abstracción del concepto de las actividades particulares realizadas en un turno de modo de permitir extensibilidad en cuanto a la ampliación en la gama de eventos posibles en un turno. La misma especifica operaciones genéricas a implementar para la obtención de datos relativos a los horarios, datos propios y comunes a todos sus subtipos, así como datos fundamentales en la asignación de recursos.

B.3.2.1.6. Viajes

Representa el concepto de viaje, independientemente del tipo, buscando aislar los atributos comunes como ser *origen, destino, horaSalida y horaLlegada*, anticipando así la posibilidad de futuros subtipos.

Se incluyen dos implementaciones, clases concretas de la misma, como son *Activo* y *Expreso*, que representan los tipos de viajes relevados y utilizados, correspondiéndose con *active* y *deadhead trips* respectivamente.

Cabe mencionar que para los *Activos* se puede especificar la preferencia en cuanto a categorías de vehículos a realizarlo. Para ello, se completa el atributo *prefCategorias* en orden prioritario descendente para ser tenido en cuenta por el

algoritmo el cual buscará, siempre y cuando sea posible, realizar la asignación a la categoría más deseada. Esto surge como un requerimiento relevado para la empresa en particular pero muestra estar presente en las semejantes.

B.3.2.1.7. Clase *Descanso*

Representa los descansos que podrán ser requeridos, o no, en cada turno según lo defina el problema. Cada uno de ellos podrá especificar distintas duraciones dependiendo de las reglas de descanso definidas en el archivo de configuración correspondiente.

B.3.2.1.8. Clase *Vehiculo*

Cada instancia de esta clase representa a una única unidad existente en la flota disponible para la cual se especifica un identificador (*id*), el identificador del depósito al que pertenece y el identificador de la categoría a la cual pertenece, la cual deberá pertenecer al rango de las especificadas como parámetros del problema.

B.3.2.2. Funcionalidades

Este módulo, quien orquesta el sistema integrado PG42 (en conjunto con los módulos *WorkerRuleModule*-WRM y *ExtensionModule*-EM) cuenta con un único caso de uso principal que consta de la ejecución del algoritmo y resolución del problema, pudiéndose configurar para especificar distintos tipos de uso. Dejando de lado las distintas funcionalidades de inicialización de soluciones, que en sí extienden la anterior, podemos categorizar distintos problemas de ejecución a ser soportados por el módulo.

- SDVSP : Single Depot Vehicle Scheduling Problem
- MDVSP: Multiple Depot Vehicle Scheduling Problem
- IDVSP: Integrated Duty Vehicle Scheduling Problem

Pudiendo agregar opcionalmente restricciones sobre categorías de viajes y vehículos, reglas embebidas, y reglas definidas.

Como funcionalidad de integración con la interfaz de usuario, se extienden las funciones de despliegue de la solución generada por el sistema respetando el formato definido de modo que pueda ser interpretada por el módulo de interfaz gráfica.

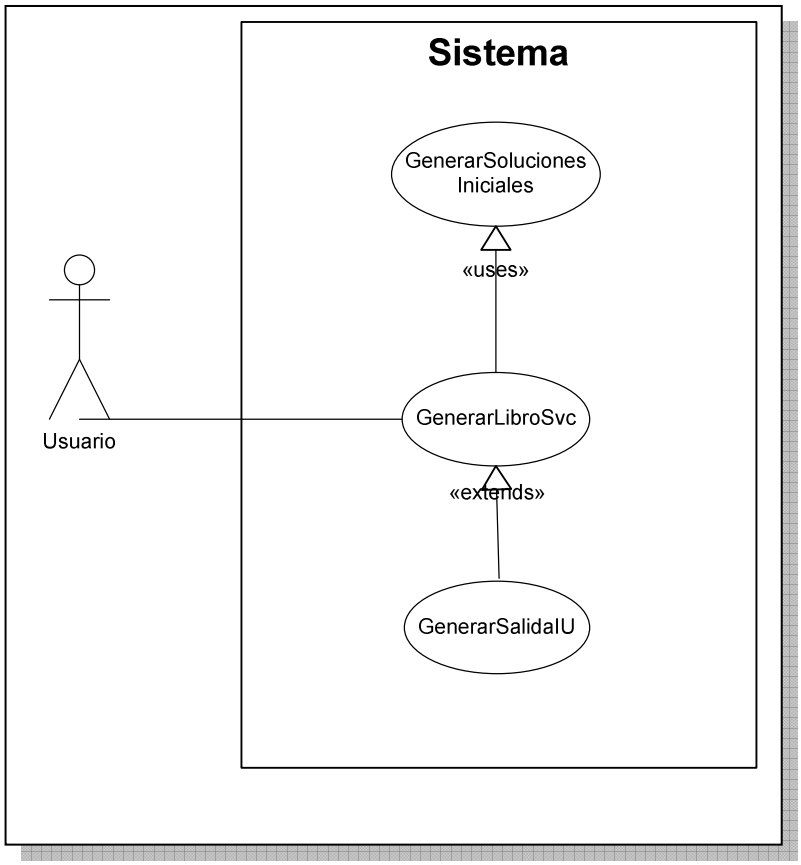

Fig. B.3.2.1. Diagrama de Casos de Uso del Módulo GAM.

B.3.2.3. Algoritmo y Motor

El diseño realizado para cumplir con los requerimientos y funcionalidades correspondientes al algoritmo y motor de ejecución esta ligado al esqueleto genérico de un algoritmo genético, particularmente como el que se muestra en la Figura B.3.2.2 que incluye llamadas al módulo de reglas laborales (WRM), integrando así las reglas definidas y manteniendo la estructura del mismo.

```
Solucion run(){
      InicializarConWRM(P(0));
      generacion=0;
      mientras (noCriterioParada) hacer
            Evaluar (P(generacion));
            Padres = Seleccionar(P(generacion))
            Hijos = AplicarCruzamientoConWRM(PadreS)
            Hijos = AplicarMutacionConWRM(Hijos)
            NuevaPob = Reemplazar(Hijos, P(generacion))
            generacion++;
            P(generacion) = nuevaPob;
      fin;
      return mejorSolucion(P);
}
```

Fig. B.3.2.2. Pseudocódigo del algoritmo integrando reglas laborales (WRM).

B.3.2.3.1. Clase *Solver*

Esta clase es la encargada de especificar el patrón de resolución correspondiente y mantiene la información actualizada del estado de exploración cuando se ejecuta de manera de aplicar los operadores correspondientes y resolver la evolución entre generaciones. Es en sí, el núcleo del motor de ejecución, ya que orquesta el proceso de resolución cuya ejecución se lleva a cabo mediante el método *run()*.

La clase concreta *Solver_Seq*, extiende a la *Solver* de modo de proveer una ejecución secuencial del algoritmo, pudiéndose incluir otro tipo de modalidades de ejecuciones como ser en paralelo, lo cual está previsto por la propia biblioteca MaLLBa a través de su implementación de MPI (*Message Passing Interface*), cuyos detalles pueden encontrarse en el anexo D.

B.3.2.3.2. Clase *Population*

Representa al conjunto de individuos que forman la población de una generación determinada. Entre sus métodos principales se encuentran la inicialización (*initialize()*), la obtención de su valor de *fitness* calculado (*fitness()*) y el método que define cómo realizar la evolución entre generaciones (*evolution()*).

B.3.2.3.3. Clase *SetUpParams* y *Operator_Pool*

La clase *SetUpParams* es instanciada por el Solver del algoritmo durante su inicialización y contiene los parámetros y valores propios del algoritmo y motor de ejecución, como ser el tamaño de la población, la cantidad de ejecuciones independientes que se desean realizar, entre otros.

Entre los parámetros contenidos en esta clase, se encuentran algunos propios de la ejecución a realizar cuyos valores son especificados por el usuario a través de un archivo de configuración *(pg42GA.cfg*):

- *independent_runs:* que representa la cantidad de ejecuciones independientes a realizar para una instancia del problema.
- *nb_evolution_steps:* es la cantidad de generaciones que deben realizarse.
- *population_size*: tamaño de la población en cantidad de individuos.
- *population_additional_size*: tamaño de los nuevos individuos (hijos) en cada generación.
- *combine*: valor booleano que habilita o no la combinación de padres e hijos para la selección.

También contiene el conjunto de operadores (*Operator_Pool*) a utilizar para el método de evolución (*evolution()*) de la población que, en este caso, serán los operadores genéticos de selección, cruzamiento y mutación, que se detallan a continuación.

B.3.2.3.4. Clase *Selection*

Permite la especificación de los operadores genéticos de selección a través de la implementación de la operación *select_one()*. Se incluyen tres tipos (clases) distintos que implementan tres de las técnicas más utilizadas:

- *Selection_Roulette_Wheel:* implementa el método de selección conocido como Rueda de Ruleta.
- *Selection_Tournament:* implementa el método de conocido como Torneo Estocástico.
- *Selection_Rank:* implementa el método de selección conocido como Selección por Ranking.

B.3.2.3.5. Clase *Crossover*

Implementa el operador de cruzamiento en el método de la operación *cross()* la cual toma como entrada los individuos a cruzar (*parents)* y en caso de ser posible, basado en las restricciones de ordenamiento y laborales, retorna los hijos (*offsprings)* generados por dicho método.

B.3.2.3.6. Clase *Mutation*

Análogamente al cruzamiento, esta clase implementa el operador de mutación. Su operación *mutate()* toma como entrada el individuo a mutar y, en caso de ser posible, basado en las restricciones de ordenamiento y laborales, retorna la nueva solución mutada.

B.3.2.3.7. Clase *Statistics* y *UserStatistics*

Ambas clases permiten la captura de valores de interés, así como la generación de estadísticas propias de la(s) instancia en ejecución, los cuales son almacenados como campos del atributo *stat* y *user_stat* respectivamente. Dichas estadísticas no solo son de interés para poder realizar calibración y ajuste para un problema determinado, sino que permiten la obtención de información valiosa que podrá tener simplemente fines informativos sobre el proceso y soluciones generadas o también podrá ser utilizada para la mejora, implementación e incorporación de nuevas técnicas que saquen provecho de dichos datos. Particularmente se utilizaron valores de *fitness* para desarrollar la técnica denomina *ajuste dinámico de la probabilidad de mutación* detallada en la sección 3.5.

B.3.3. Diagrama de Clases de Diseño

El diagrama de la Figura B.3.3.1 ha sido simplificado para facilitar su legibilidad, con el objetivo de mostrar fundamentalmente las funciones y las relaciones principales existentes entre los componentes que forman al GAM.

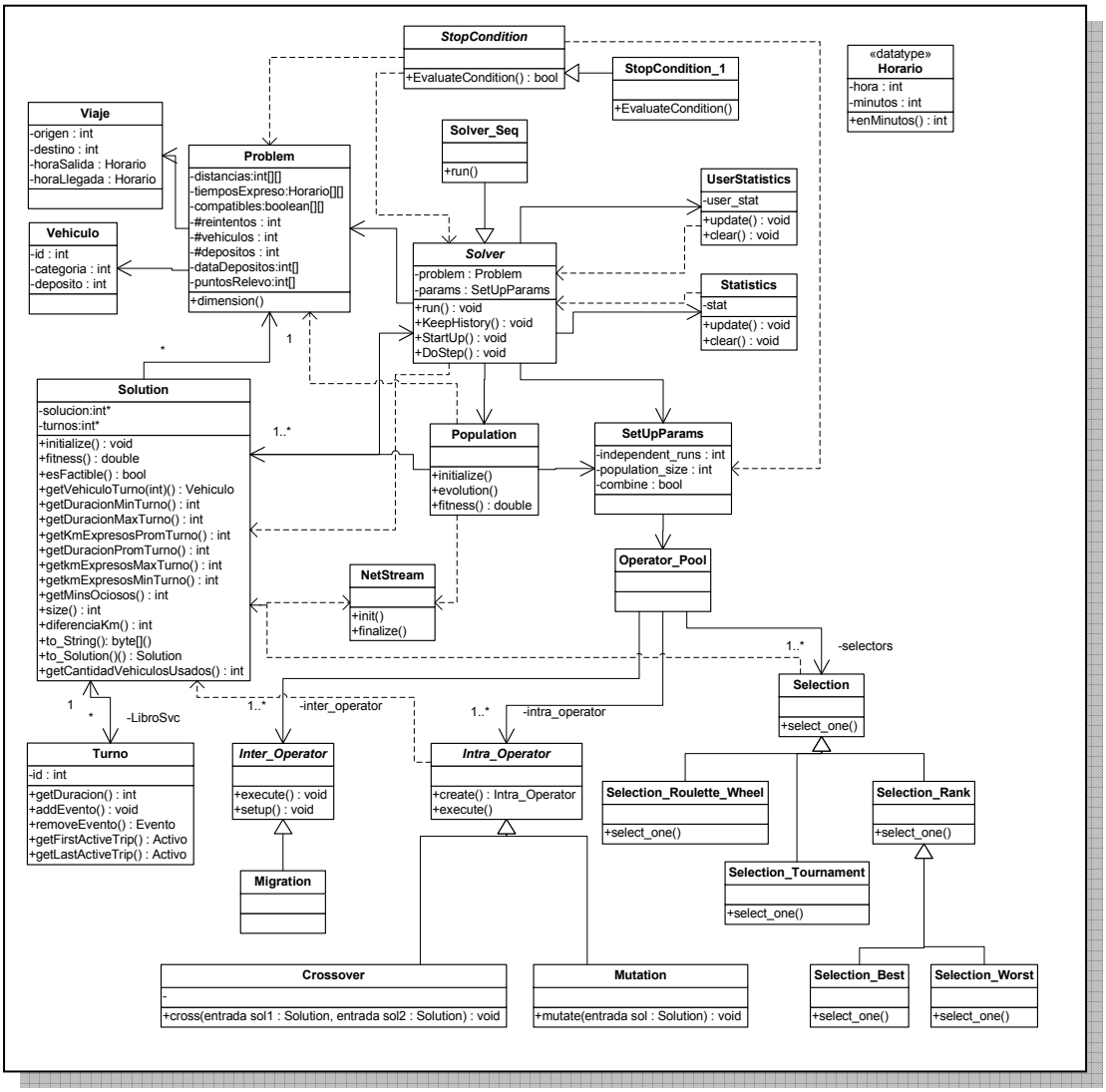

Fig. B.3.3.1 Diagrama de Clases de Diseño del GAM

B.3.4. Integración de Módulos

Para poder integrar las restricciones laborales a los distintos procesos y etapas del algoritmo se obtiene y accede a la interfaz provista por el módulo de reglas laborales (WRM) *WorkerController* cuyo detalle se especifica en el anexo C. Dado que se busca una interacción constante con este módulo que permita la generación y validación de turnos y soluciones en cada uno de los procesos principales, procesos y clases como los operadores genéticos (*Crossover y Mutation)* así como los métodos de inicialización, tendrán visibilidad sobre dicha interfaz.

También se anticipa la posibilidad de inclusión de extensiones para la cual de manera análoga, se accederá al Módulo de Extensiones (EM), obteniendo y ejecutando las extensiones definidas. Los detalles sobre estas funcionalidades e integración con este módulo se especifican en el anexo E.

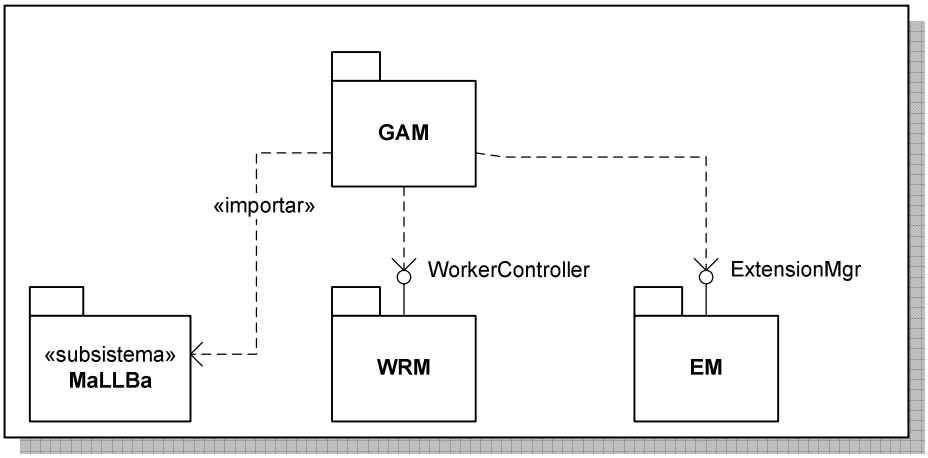

Fig. B.3.4. Diagrama de Paquetes - Integración de reglas y extensiones.

B.3.5. Diagramas de Interacción

A continuación se presentan diagramas de comunicación[UML] de las operaciones principales de este módulo.

B.3.5.1. Comunicación de Ejecución del Algoritmo

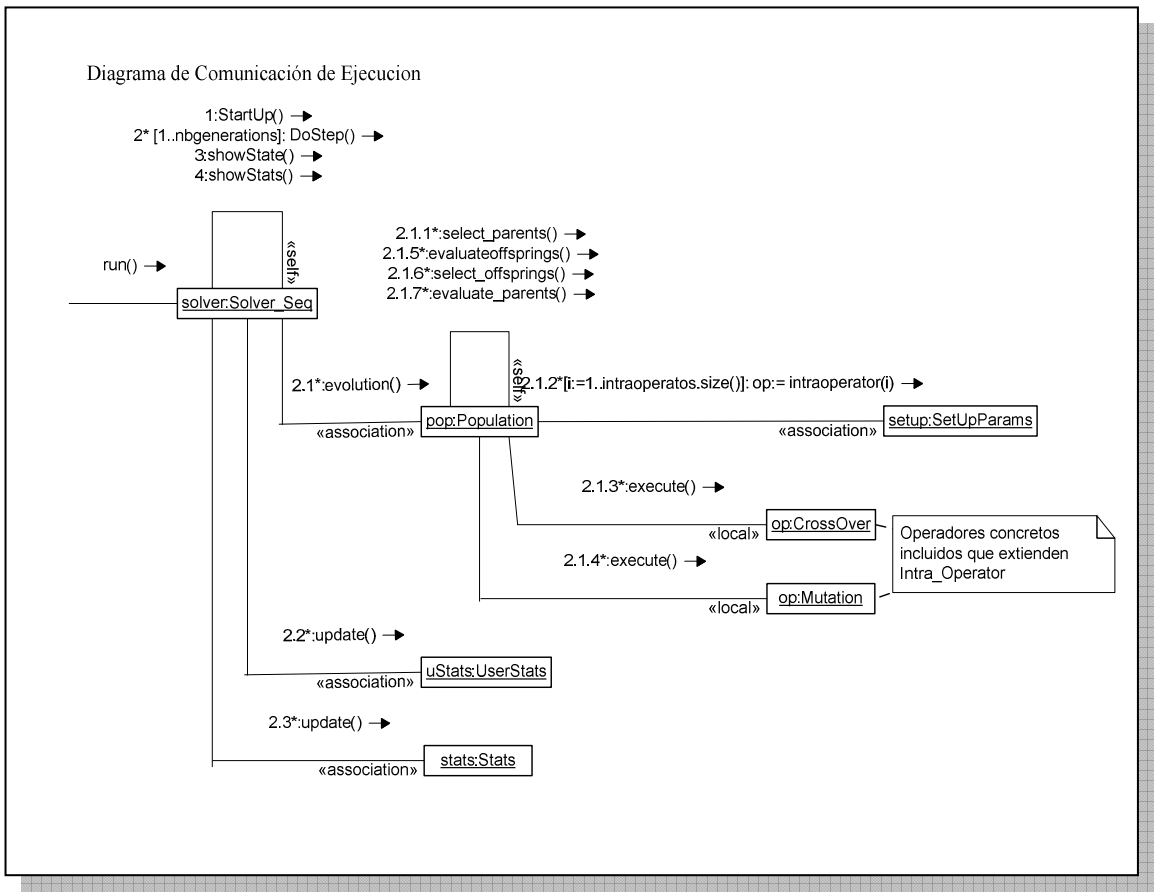

Fig. B.3.5.1.Diagrama de Comunicación de Ejecución del Algoritmo.

B.3.5.2. Comunicación de Inicialización de Solver

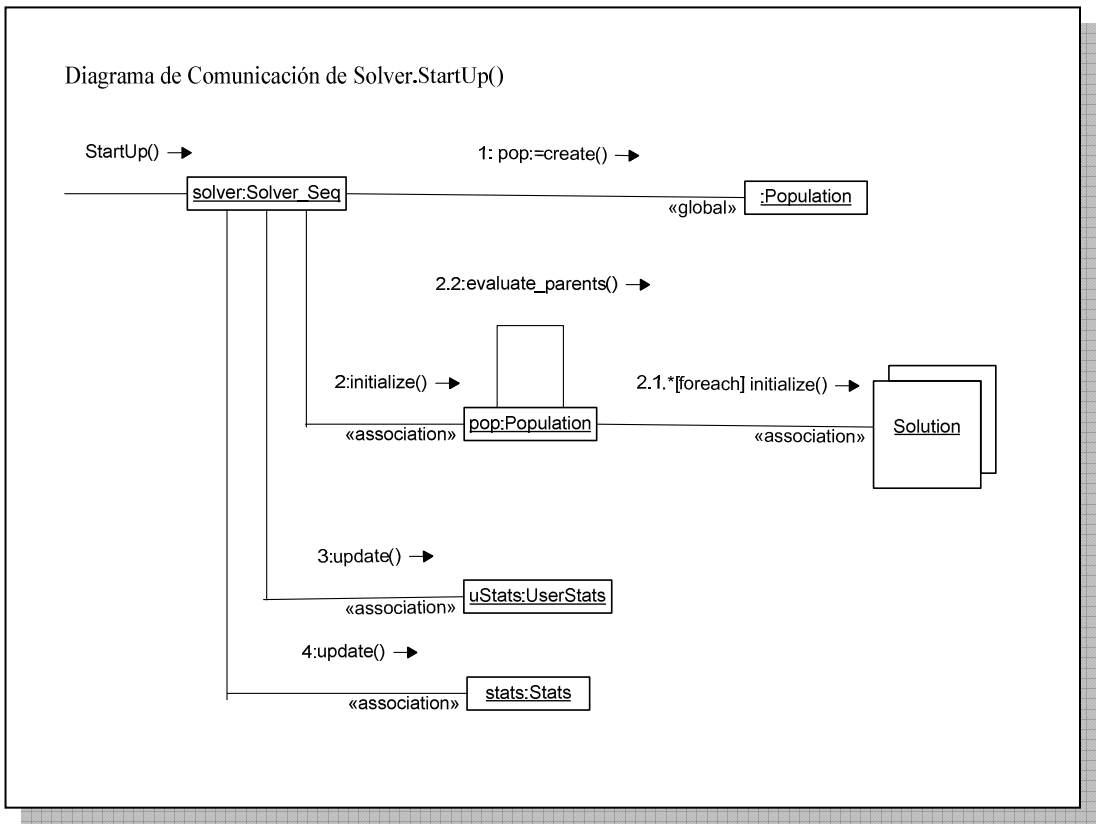

Fig. B.3.5.2.Diagrama de Comunicación de Inicialización de Solver.

B.4. Pruebas de Verificación y Validación

B.4.1. Dominio y Operadores

Estas pruebas pretenden principalmente verificar la correcta creación de la instancia del problema y soluciones así como la generación de todas las estructuras vinculadas al modelo y algoritmo para la resolución del mismo. La prueba consta de la ejecución de un caso que permitirá también la corroboración de la ejecución de los métodos del algoritmo genético como ser inicialización, generación, evolución y las técnicas empleadas en los propios operadores genéticos. Para este caso, se considerarán únicamente las restricciones propias del problema de ordenamiento, no integrando aún ninguna restricción relativa a las normas laborales, ni optimizaciones, las cuales serán realizadas luego.

Los parámetros de entrada seleccionados así como la especificación de esta instancia se encuentran en los archivos *CasoValidacion.cfg* y *CasoValidacion.txt* respectivamente. Básicamente el escenario planteado para esta prueba consta de 3 nodos, 30 vehículos distribuidos en 3 categorías y 66 viajes a cubrir que cuyos horarios se incluyen dentro del rango [5hs-21hs]. En este caso, para la función de *fitness* (*z*) se

incluyen los siguientes factores: cantidad de kilómetros expresos (f_1), cantidad de vehículos utilizados (f_2) y la diferencia entre los kilómetros realizados por cada uno de ellos y el promedio general (f_3-balance de carga), siendo el primero (*kmExpresos*) el de mayor preponderancia con un coeficiente de 0.8.

Se realizan 8 ejecuciones del módulo, cada una para 10 ejecuciones independientes consecutivamente, evaluando así un total de 80 ejecuciones independientes, obteniendo soluciones factibles para cada una de estas, cuyas asignaciones se detallan en el anexo H.

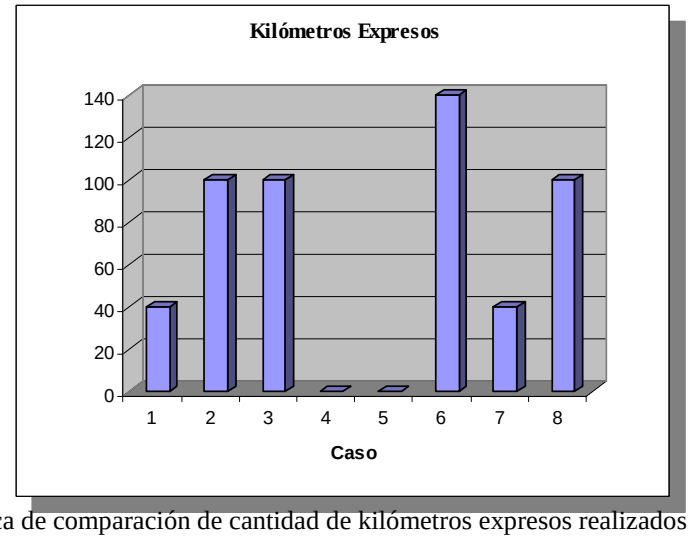

Fig. B.4.1.2. Gráfica de comparación de cantidad de kilómetros expresos realizados en cada caso.

Se propone la variación de los valores probabilísticas de los operadores genéticos de manera de realizar distintas combinaciones.

Selección	P(Cruz)	P(Mut)	Fitness	Tiempo(ms)
RR	0,6	0,01	40	3,51248E+06
RR	0,6	0,1	100	3,44035E+06
RR	0,3	0,01	100	3,13382E+06
RR	0,3	0,1	0	3,18335E+06
Torneo Est.	0,6	0,01	0	3,43610E+06
Torneo Est.	0,6	0,1	140	3,42513E+06
Torneo Est.	0,3	0,01	40	3,16482E+06
Torneo Est.	0,3	0,1	100	3,10771E+06

Fig. B.4.1.1.Tabla de valores para operadores genéticos.

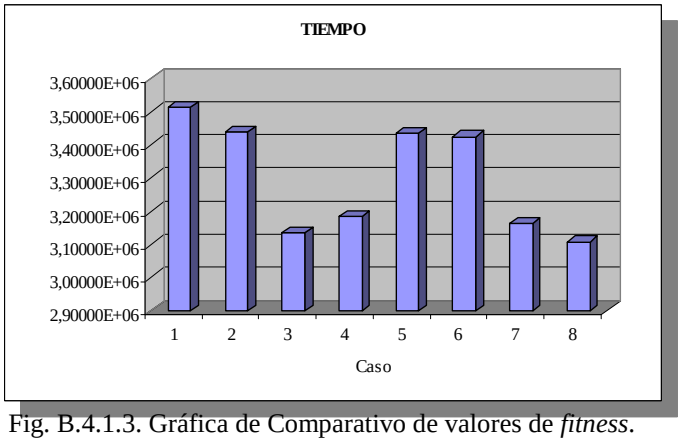

Fig. B.4.1.3. Gráfica de Comparativo de valores de *fitness*.

Dado que la inicialización consta de 3 estrategias distintas, como se detalla en la sección 3.2, se realiza el análisis de los resultados obtenidos para cada una de las estrategias por separada obteniendo los siguientes resultados para el caso presentado.

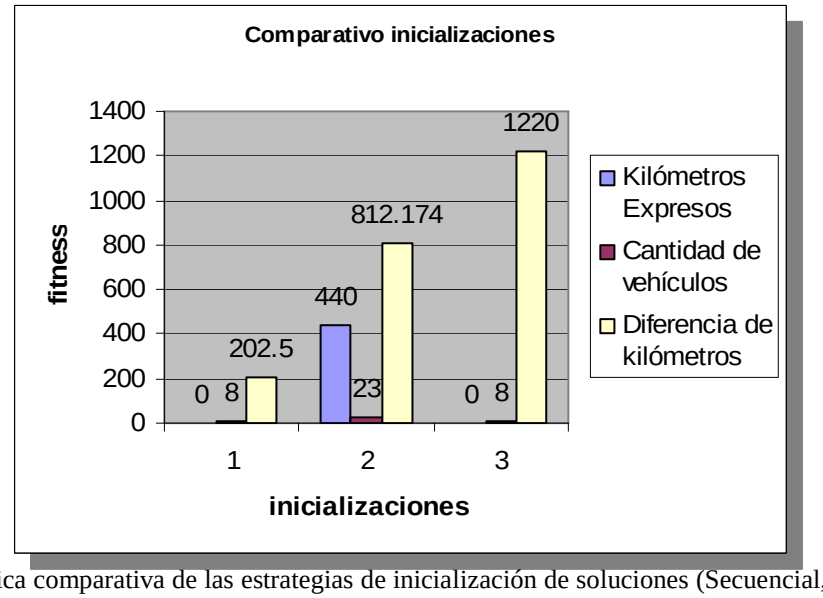

Fig. B.4.1.3. Gráfica comparativa de las estrategias de inicialización de soluciones (Secuencial, Aleatoria, Entronques)

Considerando que la función de *fitness* se plantea como multiobjetivo, siendo una combinación lineal de múltiples factores, se realiza un estudio comparativo variando lo factores que la componen individualmente a través de la ponderación de sus coeficientes (α, β, χ).

- $z(x) = \alpha.f_1(x) + \beta.f_2(x) + \chi.f_3(x)$

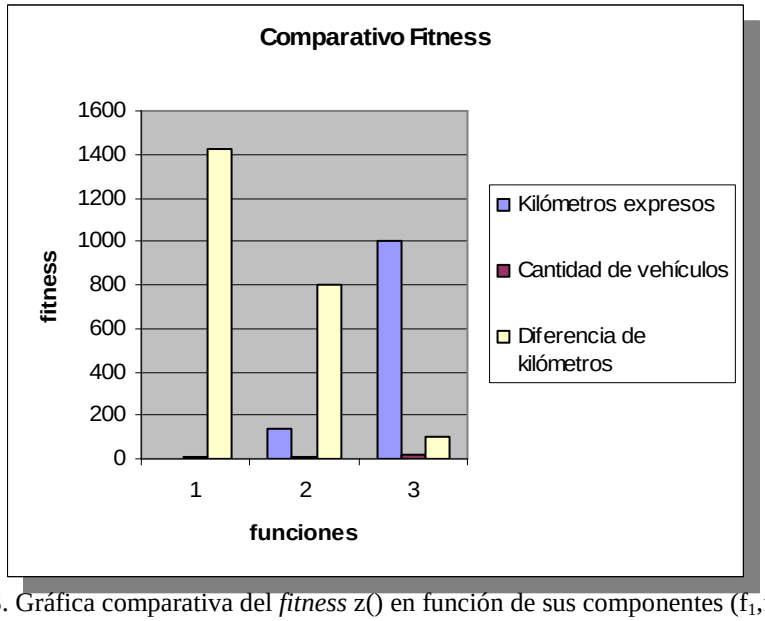

Fig. B.4.1.3. Gráfica comparativa del *fitness* z() en función de sus componentes (f_1,f_2 ,f_3)

Como podemos ver, y de acuerdo a lo esperado, cuando tomamos en cuenta solamente los kilómetros expresos (f_1), obtenemos el mejor resultado en dicha función. Y pasa lo mismo con las demás funciones. Aquí, vale la pena notar el rango de cada una de las funciones: Los kilómetros expresos, van entre 0 y 1000, la cantidad de vehículos oscila entre 6 y 14, y la diferencia de kilómetros, entre 100 y 1421. De estos valores, podemos realizar una normalización para obtener los valores de los coeficientes por los cuales multiplicar cada una de las funciones para que tengan valores que se puedan comparar de acuerdo a los intereses deseados.

B.5. Trabajo Futuro

B.5.1. Ejecución Distribuida.

Una de las ventajas de la inclusión y uso de la biblioteca MaLLBa y MPICH[MPICH] en dicho módulo abre la puerta a una funcionalidad muy atractiva en cuando a ejecución de algoritmos para problemas *complejos*, como lo es este caso. El diseño y modelo obtenido así, como el motor de ejecución del algoritmo, anticipan la posibilidad de poder configurar el motor para poder ejecutar el módulo en paralelo como un sistema distribuido en múltiples nodos admitiendo configuraciones LAN como WAN, cuyas operaciones y métodos se encuentran implementados en las clases correspondientes (*Solver_LAN y Solver_WAN*) que extienden *Solver* permitiendo así la orquestación de este tipo de ejecuciones.

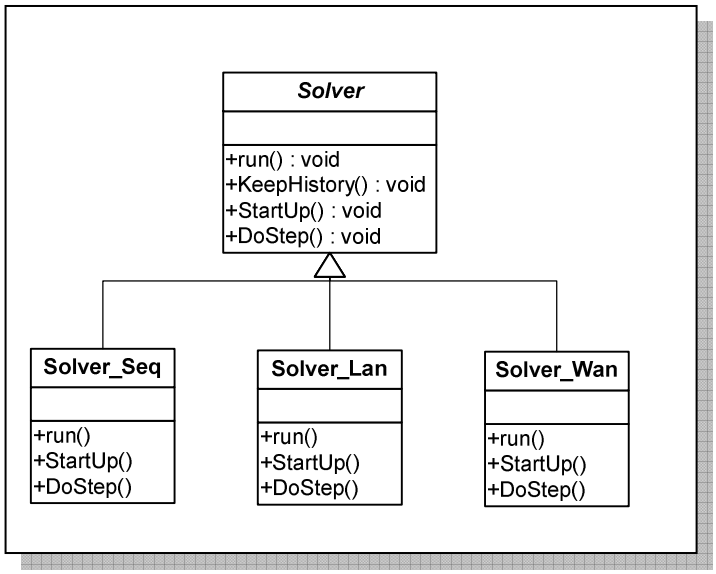

Fig. B.5.1.1.Extensiones de la clase *Solver, Solver_Lan y Solver_Wan* para ejecución en paralelo.

Dicha funcionalidad permitiría disminuir los tiempos de respuesta mediante la distribución de la carga en múltiples nodos o poder aumentar los parámetros del algoritmo buscando mantener incrementos en tiempos de respuesta acotados, con el objetivo de poder explotar mejor la técnica, en particular la evolución que implica la ejecución de un algoritmo genético para poblaciones más numerosas o mayor cantidad de generaciones.

Alguna de las pruebas y mejoras obtenidas de este tipo de ejecuciones se detallan en *Mallba: A library of skeleton for combinatorial optimisation* *[MALLBAOP]* donde se analizan las ventajas en tiempo de ejecución para una instancia del problema de asignación de recursos (*Resource Allocation Problem*[RAP]) donde se prueba una buena escalabilidad en el incremento de unidades de procesamiento.

Stages-States	Sequential time (s) on fastest machine	Speed-up			
		2 procs. 700 MHz	4 procs. 700 MHz	4 procs. 700 MHz 4 procs. 500 MHz	4 procs. 700 MHz 9 procs. 500 MHz
1000-2000	457.79	1.97	3.92	4.12	6.01
1000-2500	714.87	1.98	3.94	4.30	6.02
1000-4000	1828.22	1.99	3.96	4.31	6.41
1000-5000	2854.04	1.99	3.97	4.24	6.42
1000-7000	5594.74	1.99	3.97	4.22	6.41
1000-10000	11422.60	1.98	3.97	4.18	6.38

Fig. B.5.1.2 Resultados para el problema RAP utilizando un esqueleto MaLLBa en una red de 13PCs (4 AMD K6 700Mhz y 9 AMD K6 500Mhz)

B.6. Referencias

[MaLLBa] MaLLBa - http://neo.lcc.uma.es/mallba/easy-mallba/html/mallba.html

[OOP 07] Object Oriented Programming, http://math.hws.edu/eck/cs124/downloads/OOP2_from_Univ_KwaZulu-Natal.pdf

[OOPA 03] Object Oriented Paradigm, http://www.cs.utep.edu/sroach/S10-5380/TheObjectOrientedParadigm.PDF

[UML] Unified Modeling Language - http://www.omg.org/

[MPICH] MPICH, Implementacion MPI - http://www.mcs.anl.gov/research/projects/mpi/mpich1/

[MALLBAOP] MaLLBa: A library of skeletons for combinatorial optimization, www.lcc.uma.es/~eat/pdf/europar2.pdf

[RAP] *Enumerative Approaches to Combinatorial Optimization, Part II*. Annals of Operations Research. Volume 11, 1-4, T. Ibaraki (1988)

C. Módulo de Reglas Laborales (WRM)

C.1. Introducción

Las normativas laborales son variables con el tiempo ya que están en juego los derechos de los trabajadores y las exigencias del empleador. Dada esta variación, surge la necesidad de contar con una herramienta que permita la definición de manera sencilla de las normativas de forma que se integren como restricciones en la búsqueda de una posible solución próxima al óptimo.

El módulo de reglas laborales nos permite declarar reglas de manera simple, que se convertirán luego en restricciones del problema, así como también, dado el conocimiento que posee, será el encargado de la generación de turnos.

Este documento incluye los fundamentos, detalles y aspectos técnicos de las estrategias, decisiones de diseño y funcionalidades implementadas en este módulo.

C.2. Objetivos y Requerimientos

C.2.1. Objetivos

Este módulo cuenta con dos objetivos fundamentales que forman parte de la búsqueda de soluciones factibles al problema integrado.

El primero de ellos es permitir la definición de reglas laborales (*Worker Rules*) de manera sencilla para que éstas sean utilizadas como restricciones en la búsqueda de la solución óptima del problema. Para la definición de las mismas se utiliza una gramática (detallada en la sección C.3.1.5) que permite utilizar funciones, definición de identificadores, etc.

El segundo objetivo principal es la responsabilidad de la generación de los turnos laborales. Para esto se basa en la validación de las reglas y además tiene el conocimiento, para poder generar turnos más eficientes, de cierto comportamiento frente a reglas que se encuentran embebidas dentro de la lógica del armado de turnos.

A su vez, también es responsable de la creación de las reglas de descanso (*Break Rules*) que se podrán definir y serán utilizadas para la búsqueda de una solución.

También es el encargado de manejar la definición de las constantes relativas a lo laboral que son utilizadas por el algoritmo al generar los turnos.

C.2.2. Requerimientos

Los requerimientos principales de este módulo pueden resumirse en:

- Definición y creación de reglas laborales.
- Definición y creación de reglas de descanso.
- Definición y manejo de constantes utilizadas para la creación de turnos.
- Generación de turnos laborales basados en las restricciones definidas.
- Creación eficientes de turnos laborales basados en reglas embebidas.
- Manejo de librerías para la definición y extensión de reglas y nuevos componentes.

C.3. Funcionalidades

C.3.1. Lenguaje

Para la especificación de las reglas laborales se definió una gramática (sección C.3.1.5) que permite definir conceptos de forma tal que, puedan ser interpretados de manera correcta para la validación de las soluciones, y a su vez que sea fácil de utilizar para una persona que no tenga grandes conocimientos técnicos avanzados en programación.

El lenguaje de la misma está conformado básicamente por componentes lógicos de primer orden, que incluyen operadores de comparación, funciones, identificadores y parámetros.

Es importante aclarar que para cada elemento de la gramática se definió una sintaxis, la cual se detalla en la figura C.3.1.5, que debe ser respetada de manera de realizar las comprobaciones de factibilidad correctas en el momento de evaluar una solución.

C.3.1.1. Componentes Lógicos

Los componentes lógicos que están definidos en esta gramática son operadores de primer orden, ya sea binarios como OR y AND, y el operador unario NOT.

Para la definición de un operador lógico se define la siguiente sintaxis:

- OR(*r1*,*r2*)
- AND(*r1*,*r2*)
- NOT(*r1*)

Donde *r1* y *r2* representan otros componentes a ser evaluados. Cualquiera de ellos deberá, al ser evaluado, devolver un resultado *booleano* (*true* o *false*) para que la evaluación del operador lógico sea posible.

Cabe mencionar que el resultado de la evaluación particular de cualquiera de estos operadores lógicos es exactamente la que responde a la definición del mismo.

A continuación, se muestran las tablas de verdad donde se definen las posibles combinaciones y los resultados que se devolverán frente a la evaluación del operador.

Valor de r1	Valor de r2	Valor del OR
False	False	False
False	True	True
True	False	True
True	True	True

Fig. C.3.1.1.1. Tabla del operador OR

Valor de r1	Valor de r2	Valor del AND
False	False	False
False	True	False
True	False	False
True	True	True

Fig. C.3.1.1.2. Tabla del operador AND

Valor de r1	Valor del NOT
False	True
True	False

Fig. C.3.1.1.3. Tabla del operador NOT

La evaluación de los operadores binarios, AND y OR, se realiza por circuito corto, es decir, que podemos tener el resultado de la evaluación sin la necesidad de evaluar la condición entera. Casos como estos, los podemos ver en la filas 3 y 4 de la Figura C.3.1.1.1, o las filas 1 y 2 de la Figura C.3.1.1.2, ya que el valor de la evaluación del primer operando determina el valor de la evaluación completa. Esto permite que el programa sea más eficiente al momento de la evaluación de las condiciones.

C.3.1.2. Operadores de comparación

Los operadores de comparación que están definidos en la gramática son los operadores clásicos de comparación de elementos: mayor (GREATER), menor (LESSER) e igual (EQUAL).

Para la definición de un operador de comparación se definió la siguiente sintaxis:

- EQUAL(*r1*,*r2*)
- GREATER(*r1*,*r2*)
- LESSER(*r1*,*r2*)

Donde *r1* y *r2* representan un componente a ser evaluado. Cualquiera de ellos deberá, al ser evaluado, devolver solamente un vector de valores comparables para que la evaluación del operador de comparación sea válida. Debido a esto, surgen 4 alternativas de posibles evaluaciones, las cuales son dependientes de la cantidad de elementos devueltos en cada vector. Los casos soportados de acuerdo a la cardinalidad son: 1x1, Nx1, 1xN y NxN. Para los casos Nx1 y 1xN, el resultado de la evaluación será la conjunción de los resultados de cada evaluación individual de los N elementos con el elemento del vector de dimensión 1. En ambos casos se utiliza la evaluación de circuito corto al igual que en la evaluación de los operadores lógicos. Sin embargo, para el caso NxN, la evaluación es diferente. Se devuelve la conjunción de los resultados de la evaluación del elemento i de cada vector. A continuación se muestra un pseudocódigo de cada caso para aclarar la lógica de cada evaluación.

Caso 1x1:

```
operadorComparacion (valor A, valor B) {
      return comparación (A, B)
}
```

Caso Nx1:

```
operadorComparacion (vector A, valor B) {
      itero en cada posición i del vector A {
            si comparación (A[i], B) NO es válida
                  return resultado falso
      }
      return resultado verdadero
}
```

Caso 1xN:

Este caso es totalmente análogo al caso anterior.

Caso NxN:

```
operadorComparacion (vector A, vector B) {
      itero en cada posición i de ambos vectores {
            si comparación (A[i], B[i]) NO es válida
                  return resultado falso
      }
return resultado verdadero
}
```

C.3.1.3. Funciones

A pesar de contar con operadores lógicos y de comparación que permiten, junto con los identificadores, definir muchas reglas, se decide brindar la posibilidad de contar con funciones como reglas a evaluar. Esto permite crear un módulo de reglas extensible, ya que si se quiere crear alguna regla cuya condición sea más que una evaluación lógica de conceptos conocidos o comparación de valores, no se podría lograr. En cambio, con la introducción de este concepto logramos que cualquier regla, que no sea posible de especificar con operadores lógicos o comparativos, sea posible de implementar mediante una función.

En la sección C.3.3 se explica el funcionamiento de la validación de reglas, pero es importante destacar, cómo se crea una función y cómo se puede utilizar.

La manera correcta de definir una función, cuyo nombre puede ser cualquiera a excepción de los nombres reservados para los componentes lógicos y operadores ya definidos, considerados como palabras claves, es:

nombreDeLaFuncion()

C.3.1.3.1. Clase de ejemplo

A continuación se muestra un ejemplo de cómo incorporar una clase que representa una función y que será evaluada dentro de las posibles reglas laborales que condicionarán la factibilidad de las soluciones buscadas.

```
class f_A : public Function {

public:

      f_A () {
            sprintf (this->className, "%s", "f_A");
            this->paramQuantity = 0;
      }

      virtual ~f_A() {}

      virtual bool isFeasible(Solution* sol) const {
            return true;
      }

};
```

Fig. C.3.1.3.1. Definición de una función particular

En este ejemplo se puede ver que basta con definir una clase que extienda *Function*, con su respectivo constructor y destructor, y que tenga implementado el método *isFeasible* para que pueda ser evaluada como regla.

Es importante aclarar que si bien la clase se define de manera simple, a pesar que el método puede resultar muy complejo. También se debe declarar, al igual que todas las reglas, y tener definidas 2 operaciones más para que pueda ser creada como regla. En la sección G.4.1 se explica en detalles todo el proceso completo, dejando aquí simplemente a modo de ejemplo la definición de una función particular.

C.3.1.4. Parámetros

Los parámetros son una lista de valores utilizados en los operadores de comparación.

Como se puede ver en el diagrama de clases de diseño (Figura C.4.2.1), existen 3 tipos de parámetros. Por uno lado, los que extienden del concepto "*Value*", por otro los que extienden de "*MultipleValue*" y también los que extienden de "*Parameter*" pero sin extender de alguna de las 2 anteriormente mencionadas.

La principal diferencia entre las dos primeras, es que los de la primer categoría, al ser evaluados, devuelven un único valor, es decir, una lista de dimensión 1. Este valor es usado siempre para realizar las comparaciones. Por lo contrario, los que pertenecen a la segunda categoría, devuelven una lista de dimensión variable, es decir, devuelven N elementos, siendo N > 2.

Los elementos que extienden *Parameter* directamente, a priori, no se sabe cuántos elementos retornan, ya que debe existir cierta lógica programada para devolver estos valores.

A partir de estas clases, se pueden definir cualquier tipo de parámetro que se desee. Dada la flexibilidad que se desea, se pueden encontrar a los parámetros como elementos que se podrían diferenciar en dos categorías. Una podría ser la de parámetros

como valores (denominados constantes), es decir, solamente tienen un valor o conjunto de valores fijos pre establecido de antemano, y la otra sería la de los parámetros como identificadores (denominados identificadores), es decir como concepto, el cual tiene valor propio y una lógica determinada para obtener su valor o lista de valores.

A fin de esclarecer lo que se ha presentado hasta ahora, se podría decir que todos los elementos que extienden de *Value* o *Multivalue* podrían ser considerados como un valor constante mientras que todo elemento que extiende de *Parameter* podría ser considerado como un identificador.

Si vemos como categorización, la clase a la que extienden, se puede ver que los parámetros podrían ser clasificados dentro de dos categorías diferentes y totalmente disjuntas. Un ejemplo podría ser que se necesite un parámetro para el cálculo de km promedio incurridos para cada vehículo, donde sería un objeto *Parameter* que devuelve un valor calculable en tiempo de ejecución, y por otro lado, tendríamos el valor que sería con el cual se compara, y podría ser un objeto *Value* que tiene su valor definido de antemano. De la misma manera se aplica para los objetos que extiendan de *Multivalue*.

C.3.1.4.1. Clase de ejemplo

A continuación se muestra un ejemplo de cómo incorporar una clase que representa un parámetro (podría categorizarse como identificador) y que será utilizado en la evaluación de algún operador de comparación para alguna de las posibles reglas laborales que condicionarán las soluciones buscadas al problema de optimización.

```
class ParamDummy : public Parameter {

public:

	ParamDummy ()  {
		sprintf (this->className, "%s", "PARAMDUMMY");
		this->values2return = 0;
	}

	virtual ~ParamDummy()  {}

	virtual int* evaluate (Solution* sol)  {
		this->values2return = 3;
		int* retVal = new int [3];
		retVal [0] = 9;
		retVal [1] = 13;
		retVal [2] = 54;
		return retVal;
	}

};
```

Fig. 3.1.4.1.1. Definición de un identificador particular

En este ejemplo se puede ver que basta con definir una clase que extienda *Parameter*, con su respectivo constructor y destructor, y tenga implementado el método

evaluate() para que pueda ser evaluada como parámetro de cualquier operador de comparación que forme parte de una regla.

Es importante aclarar también, al igual que para las funciones, que si bien la clase se define de manera sencilla, también deben declararse otras operaciones para que el parámetro pueda ser creado y utilizado de manera correcta. En la sección G.4.1 se explica en detalle el procedimiento.

C.3.1.5. Gramática utilizada

La Figura C.3.1.5 especifica la gramática definida.

```
Gramática del Lenguaje

WorkerRule  ::=   Rule

Rule        ::=   Op_Binario(Rule,Rule)   |
                  Op_Unario(Rule)         |
                  Op_Comp(Param,Param)    |
                  FUNCION()

Op_Comp     ::=   GREATER |
                  LESSER  |
                  EQUAL

Param       ::=   IDENTIFICADOR |
                  CONSTANTE

Op_Binario  ::=   OR |
                  AND

Op_Unario   ::=   NOT
```

Fig. C.3.1.5. Gramática del Lenguaje

C.3.1.6. Definición de reglas

Para definir las reglas se cuenta con un archivo llamado *rules.xml*. El mismo debe estar ubicado en la carpeta */conf* del proyecto para que las reglas puedan ser creadas correctamente y para el posterior uso de las mismas.

Cada regla debe ser creada con el identificador *'WorkerRule'* la cuál debe estar ubicada como hija del tag *'WorkerRules'*.

En la Figura C.3.1.6 se muestra un ejemplo del archivo de configuración donde se especifican las reglas, asumiendo ID y VALUE como definidos.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Archivo de Definición de Reglas Laborales -->
<Rules>
     <WorkerRules>

           <WorkerRule>GREATER(ID,VALUE)</WorkerRule>

     </WorkerRules>

     <BreakRules>
           <BreakRule>
                 <duration>15</duration>
                 <quantity>3</quantity>
                 <start>60</start>
                 <end>420</end>
                 <gap>1</gap>
           </BreakRule>
     </BreakRules>
</Rules>
```

Fig. C.3.1.6. Ejemplo de archivo de definición de Reglas (*rules.xml*).

Como se observa en la Figura C.3.1.6, en este archivo también se definen las reglas de descanso (*Break Rules*). Este concepto será descrito en la sección C.3.2.

El valor que debe contener cada regla, es la condición que se desea imponer y debe respetar la gramática definida en la sección anterior. Para el caso de los valores predefinidos, se utiliza el siguiente formato:

- *Value*: "value"
- *Multivalue*: "{value1, value2, ..., valueN}"

Donde value y valueJ $/ J \in \{1..N\}$ son enteros.

C.3.2. Reglas de Descansos (*Break Rules*)

Las reglas de descanso son un concepto muy importante a la hora de la búsqueda de una solución óptima para el problema de optimización. Por este motivo, es que se toma la decisión de integrarla a la generación de turnos especialmente diseñada para poder asignar los descansos de manera eficiente.

Para la definición de estas, basta con agregar una nueva entrada al archivo *rules.xml* en el tag *'BreakRules'*. La misma debe estar bajo el nombre de *'BreakRule'* y debe contener los atributos definidos para que la regla de descanso pueda ser creada correctamente. Estos valores son:

- **duration**: representa la duración del descanso.

- **quantity**: indica la cantidad de veces que esta regla puede ser asignada dentro de un turno laboral.

- **start**: indica a partir de qué minuto del turno laboral puede asignarse una instancia de esta regla.

- **end**: indica a partir de qué minuto del turno laboral no puede asignarse más instancias de esta regla.

- **gap**: indica la cantidad de minutos dentro del turno laboral que son necesarios de esperar para poder asignar otra instancia del descanso.

A continuación se muestra un ejemplo de configuración de una regla de descanso. El mismo indica que deberán existir solamente tres descansos (*quantity*), cuya duración debe ser de 15 minutos (*duration*). Estos pueden ser gozados después de la primer hora del turno (*start*) y antes de la séptima hora del mismo (*end*). También se indica que luego de ser asignado un descanso, se debe esperar (dentro del turno) al menos 10 minutos (*gap*) para poder asignar el siguiente descanso. El orden de aparición de las etiquetas (*tags*) de cada regla de descanso (*Break Rules*) determinará la preferencia (descendente) de aplicación en cso que sea posible utilizar más de una regla para un mismo turno.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Archivo de Definición de Reglas
Laborales-->
<Rules>
      <WorkerRules>
            ...
      </WorkerRules>
      <BreakRules>
            <BreakRule>
                  <duration>15</duration>
                  <quantity>3</quantity>
                  <start>60</start>
                  <end>420</end>
                  <gap>10</gap>
            </BreakRule>
      </BreakRules>
</Rules>
```

Fig. C.3.2. Definición de una regla de descanso (Break Rule)

C.3.3. Validación de Reglas

La validación de las reglas se lleva a cabo mediante la ejecución del método *isFeasible()* siguiendo el patrón *composite*[UML] como muestra la Figura C.4.2.1. Es decir, que para la validación de reglas compuestas, se invoca recursivamente el método para cada uno de sus componentes.

C.3.4. Generación y Validación de Turnos

La técnica definida como estándar (*default*) para la generación de turnos, parte de la asignación de vehículos detallada en el cromosoma de la solución, y considera por procesadas las restricciones referentes al ordenamiento realizadas en el GAM. Es así que se busca realizar para cada vehículo, la generación de turnos procurando minimizar la cantidad de estos, aproximar su duración al valor *óptimo deseado* y cumplir con las reglas de descanso definidas, priorizando la aplicación de una u otra según lo indique la definición de las mismas.

Es decir, sean D el conjunto de turnos (*duties*) pertenecientes a la solución *S* y *B* el conjunto de reglas de descanso (*breakRules*) se desea:

$$\min\left\{ \#D + \sum_{d \in D} (|\, duracionOptima - duracion(d)\,|) \right\} \;/\; \forall\, d \in D, \exists\, b_i \in B \,/\, b_i.valida(d)$$

Donde *valida()* representa un predicado que indica la posibilidad de aplicación de la regla de descanso *b al turno d*. Cabe mencionar que ante la posibilidad de aplicación de distintas reglas (*b), i* representará la prioridad más alta en cuanto a preferencia definida.

El proceso de generación de los turnos, utiliza la asignación de vehículos de modo de validar el servicio de cada uno de estos. Se agrupan los viajes para formar los turnos procurando aproximarlos a la *duración óptima deseada* (dato de entrada). Durante esta etapa, cada turno generado es procesado para validar el cumplimiento de reglas de descanso y puntos de relevo. Tratándose de dos restricciones de fuerte impacto sobre la factibilidad, su temprana evaluación ahorra y reduce los tiempos y costos de ejecución de procesar soluciones, que a priori, se pueden descartar o evaluar como no factibles. El pseudocódigo de la Figura C.3.4.1 ilustra los pasos principales para la generación de servicios de cada vehículo.

```
generarTurnos(){
    for each v in Vehicles do
      while (viajesSinAsignarATurno(v) > 0 &&
              isAsignacionFactible(v))
            Turno   t= generarTurno(vehiculo);
            if (!validaBreakRules(t) ||
                  !validaReliefPoints(t))
                    regenerarTurno=true;
                  /*flag que indica la factibilidad de la
                  asignacion de v.
                  if (¡isAsignacionFactible(v)) descarta
                  sol.*/
                  solFactible = isAsignacionFactible(v);
             else
                   agregarTurnoALibroSvc(v,t)
        end;
    end;
}
```

Fig. C.3.4.1 Pseudocódigo de generación de turnos.

El proceso de validación de puntos de relevo es más simple, ya que consta de la evaluación de predicados sobre los lugares de inicio y fin de cada turno que validarán o no el servicio. Particularmente, para la realidad considerada, se contempla la posibilidad de especificación de los denominados “relevos al vuelo”, que implican la posibilidad de realizar los relevos de tripulación para un vehículo en un determinado momento de un servicio, sin necesidad que este se encuentre en un depósito (o terminal). Esto brinda mayor granularidad a la hora del armado de turnos, ya que se puede ajustar la asignación de viajes cercanos o que excedan los “límites” de duración más fácilmente, por ejemplo asignando nueva tripulación. Pese a que la técnica por defecto considera dichos puntos solamente a modo de validación, se anticipa (ofreciendo mecanismos de extensión) la posibilidad del uso de técnicas de mejora basadas en dicha información. Una de ellas, podría ser, que ante la finalización de un turno (intermedio de un servicio) en un punto que no fuera de relevo, busque alguno cercano con algún algoritmo del tipo BFS (*Breadth First Search*), y se analice la posibilidad de generación de un viaje expreso hacia el mismo, que permita la finalización del turno como corresponde (en un punto de relevo).

C.3.4.1. Reglas Embebidas

Durante el estudio del problema y relevamiento de antecedentes se pudo apreciar la existencia de conceptos propios a la realidad, cuya relevancia suele ser fundamental e independiente a una instancia particular o a la variación que pueda existir en valores o unidades consideradas. Contemplar dichos conceptos en alguna o varias etapas permiten al algoritmo hacer uso del conocimiento sobre la realidad manejada. De este modo, y teniendo en cuenta la parametrización de valores correspondiente, la posibilidad de considerar las propiedades de dichos conceptos permite modelar y aplicar el

conocimiento a las técnicas y algoritmos utilizados con el objetivo de obtener mejores resultados.

Dicho lo anterior, se propone embeber conceptos comunes a diferentes realidades, escenarios e instancias de la problemática en cuestión, de modo de poder sacarle provecho a la información de los mismos durante la etapa de ordenamiento y asignación con el objetivo de buscar soluciones teniendo en cuenta dicha información y no simplemente validar contra la misma. Esto no solo mejora la eficiencia y los costos de ejecución en muchas de las técnicas empleadas, como ser las de operadores genéticos, ya que permite la obtención de soluciones factibles de forma más sencilla, sino que también permite mejorar velozmente los resultados de la función de *fitness* de la población, lo cual debería llevar a mejores soluciones.

C.3.5. Definición de Constantes

Considerando los conceptos embebidos en el algoritmo como se ha explicado en la sección 4.3.1, existe la posibilidad de necesitar de ciertos valores para realizar comparaciones, operaciones aritméticas u otras evaluaciones. Dado que el objetivo apunta a una solución de un problema de optimización, estos valores pueden variar con el fin de explorar diferentes soluciones y poder analizar cuál es más efectiva. Por este motivo es que se incorpora la definición de constantes en el módulo.

Estas constantes tienen como fin, sustituir valores en el código pero sin tener la necesidad de compilar nuevamente. Simplemente se definen en un archivo *xml*[XML] y el módulo levanta esos valores en tiempo de ejecución para ser utilizados a demanda.

No hay restricciones con la cantidad de constantes que se pueden definir, pero lo que sí es importante de destacar es que no se pueden repetir nombres para evitar inconsistencias al momento de la resolución.

Estas constantes se deben definir en el archivo *constants.xml* situado en el directorio *conf*. Un ejemplo del mismo se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<Constants>
      <constant name=MAX_TIEMPO_TURNO value=480/>
      <constant name=MINTURNO value=300/>
</Constants>
```

Fig. 3.5.1. Archivo de definición de constantes (*constants.xml*)

Como muestra la Figura 3.5.1, cada constante queda definida con 2 valores o atributos. El primero de todos y fundamental es su nombre o identificador (*name*), el cual va a ser utilizado para definir la constante y para utilizarla dentro de la lógica. El siguiente atributo, es su valor (*value*).

En caso que se quiera utilizar una constante que no se encuentre definida en el archivo pertinente, el módulo devolverá el *String* vacío ("") al momento de invocarse la operación para pedir el valor de la misma.

En la Figura 3.5.1 vemos la definición de 2 constantes cuyos valores representarían, en minutos, la duración mínima y máxima de un turno, exceptuando horas extras. Es decir, que para este ejemplo, un turno debería durar como mínimo 5 (5*60) horas y como máximo 8 (8*60).

C.4. Diseño

C.4.1. Consideraciones Generales

Este módulo fue diseñado pensando en posibles extensiones que pueden surgir, ya sea de operadores lógicos, de comparación, o funciones.

Para facilitar la inclusión de nuevos componentes con el fin de lograr una extensión en la gramática para la definición de nuevas reglas, este módulo se diseñó de modo de permitir agrupar los componentes en grupos genéricos como ser lógicos, de comparación, etc. De este modo, mediante la extensión de alguna de las clases base, se puede incluir estos nuevos componentes.

Para lograr independencia y dinamismo en la inclusión de nuevo código fuente, se recurre a una modalidad bastante conocida, teniendo en cuenta la restricción del lenguaje, como lo es el uso de bibliotecas dinámicas (.so)[so].

C.4.2. Diagrama de Clases de Diseño (DCD)

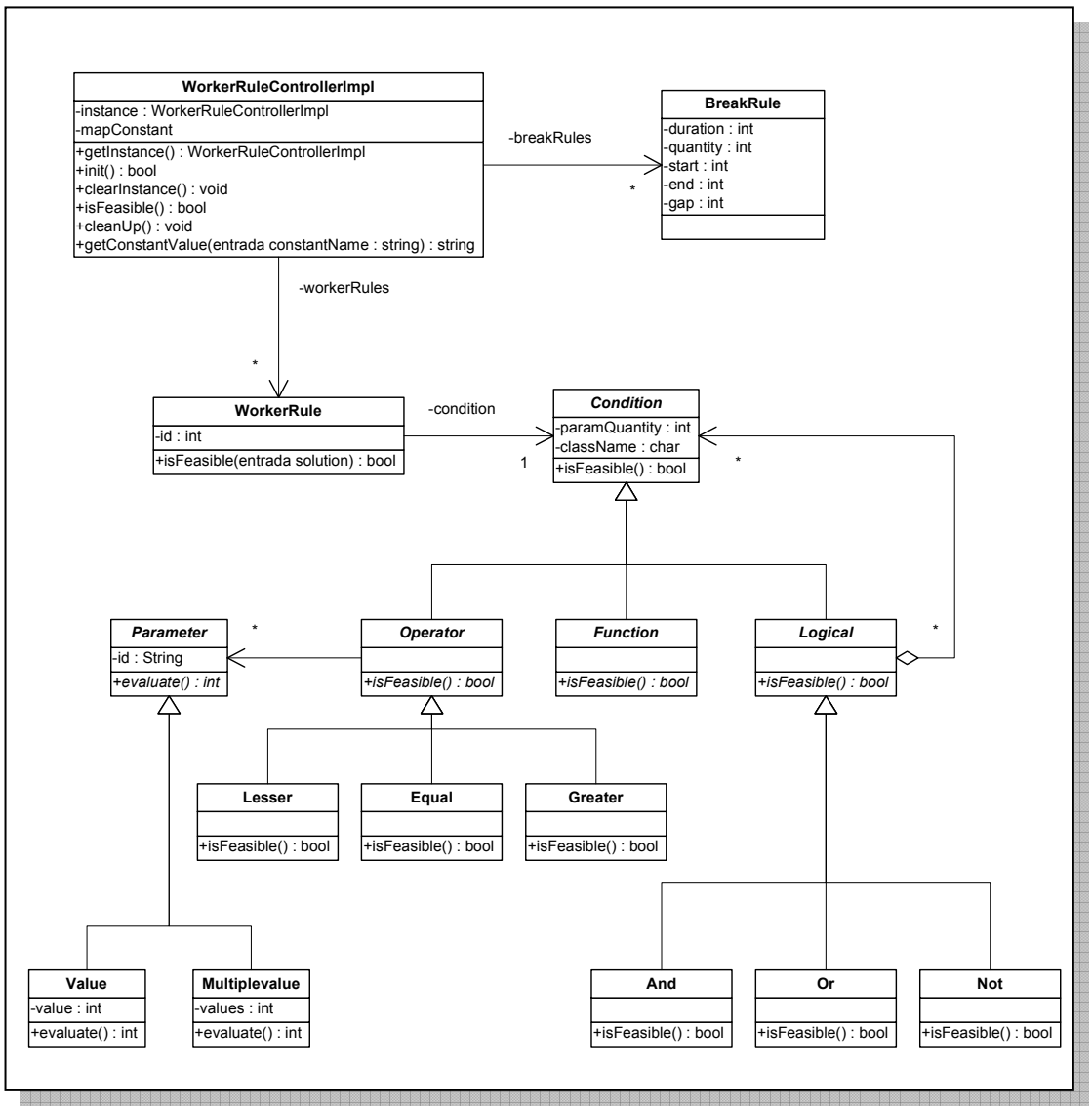

Fig. C.4.2.1. Diagrama de Clases de Diseño del WRM.

A continuación se explica cual es el objetivo de cada una de estas clases para dejar más en claro el diseño de este módulo.

WorkerRuleController: Define la interfaz con las operaciones que se expondrán del módulo y que deben implementarse para el correcto funcionamiento del mismo.

WorkerRuleControllerImpl: Esta clase implementa la interfaz anterior y es una clase que se modeló basada en el patrón de diseño Singleton[SINGLE]. La función *init* es la que se encarga de inicializar todas las estructuras y de realizar la lectura de los archivos de configuración que definen las reglas (ya sea laborales o de descanso) y las constantes, para instanciar las mismas. Por otro lado, la función *cleanUp*, se encarga de

liberar todas las estructuras que se han creado para el almacenamiento de reglas y constantes. También se encarga de destruir todas las reglas laborales y de descanso.

WorkerRule: Esta clase es la que modela una regla laboral, la cual contiene un id para identificarla y un condición que es la que se evalúa con la solución para ver la factibilidad de la misma.

Condition: Esta clase como se puede observar modela la condición de una determinada regla laboral. Es una clase abstracta por lo cual se debe extender de alguna de sus subclases para la regla se pueda evaluar.

Operator: Esta clase modela los operadores de comparación explicados anteriormente. Esta clase también es abstracta y nos permite crear nuevos operadores para poder escribir nuevas reglas que con estos operadores no se puedan lograr.

Lesser: Representa el operador MENOR, es decir que dado LESSER(A,B) devuelve si A < B.

Greater: Representa el operador MAYOR, es decir que dado GREATER(A,B) devuelve si A > B.

Equal: Representa el operador IGUAL, es decir que dado EQUAL(A,B) devuelve si A = B.

Function: Si se quiere extender nuestras reglas con cierta lógica que no se pueda representar con los operadores brindados, se puede implementar una nueva clase que extienda de esta clase, la cual tenga toda la lógica correspondiente.

Logical: Nos permite modelar cualquier operador lógico que se quiera. Es una clase abstracta y por este motivo es que si se desea algún operador nuevo, basta con extender de esta clase e implementar la función *isFeasible* con el fin de que tenga la lógica deseada.

And: Representa el operador de lógico AND. Como se explicó anteriormente, devuelve el "and lógico" de 2 resultados booleanos.

Not: Representa el operador de lógico NOT. Como ya se mencionó, devuelve la negación del parámetro booleano recibido.

Or: Representa el operador de lógico OR. Como se ha dicho anteriormente, devuelve el "or lógico" de los 2 parámetros booleanos que forman parte de la regla.

Parameter: Nos permite modelar cualquier parámetro que quiera utilizarse en los operadores de comparación. Como se explicó anteriormente, de esta clase pueden extenderse nuevos parámetros que se pueden usar como identificadores al momento de la comparación.

Value: Como ya se ha explicado, esta clase modela un determinado tipo de parámetro, el cual devuelve un único valor entero. El mismo se encuentra previamente definido en el archivo de configuración de las reglas.

Multivalue: Modela un determinado tipo de parámetro, el cual devuelve un arreglo de valores enteros. Estos valores se encuentran previamente definidos en el archivo de configuración de las reglas.

C.4.3. Diagramas de Interacción

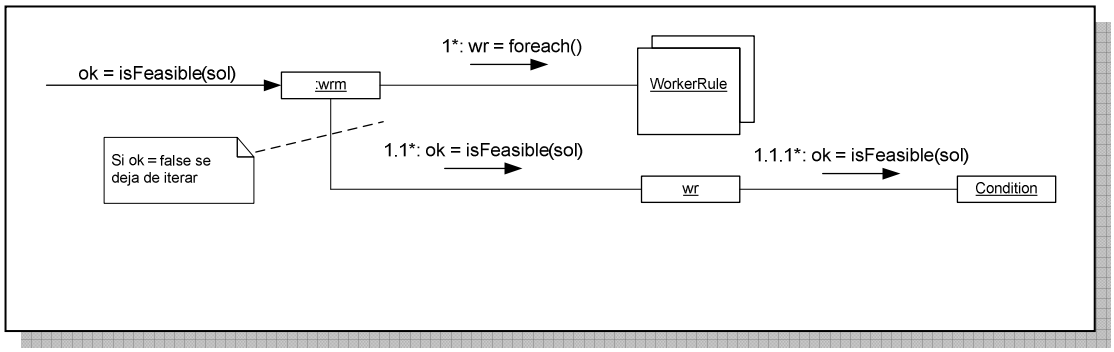

Fig. C.4.3 Diagrama de interacción de validación de reglas (*isFeasible()*)

C.5. Configuración

La configuración del módulo se realiza a través de los archivos *rules.xml* y *constants.xml*, cuyo detalle se explica en el anexo G.

C.6. Pruebas de Validación

Si bien se han realizado varias pruebas unitarias con el fin de detectar inconsistencias o errores en el módulo, las pruebas que hemos tomado como validación, han sido las pruebas de integración del sistema, la cuales nos muestran los turnos generados con las restricciones definidas.

C.7. Trabajo Futuro

C.7.1. Extensiones del Lenguaje

Como posibles extensiones se pueden mencionar la generación de nuevos operadores, ya sean lógicos o de comparación con el fin de tener un espectro mayor de posibles reglas a definir. Se propone la extensión del lenguaje utilizado para la definición de reglas a alguno más potente que permita la declaración de variables y propiedades, como ser OCL[OCL].

C.8. Referencias

[UML] Unified Modeling Language, OMG - http://www.omg.org/
[XML] Extensible Markup Language (XML) - http://www.w3.org/XML/
[so] Shared Library - http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html
[SINGLE] Singleton Pattern - http://userpages.umbc.edu/~tarr/dp/lectures/Singleton-2pp.pdf
[OCL] Object Constraint Language - http://www.omg.org/technology/documents/formal/ocl.ht

D. Biblioteca MaLLBa

D.1. Introducción

La biblioteca MaLLBa surge como un proyecto coordinado entre tres grupos de investigación radicados en MA (Málaga), LL (La Laguna, Tenerife) y Barcelona[OEGD]. MaLLBa, es un proyecto que básicamente intenta poner más cerca del usuario diversas técnicas de optimización, entre ellas las de computación evolutiva. Sus creadores, notaban que las técnicas modernas de optimización[MHCP93], se investigaban, pero no eran muy utilizadas en la práctica, es por esto que surge MaLLBa.

MaLLBa, entonces, es una biblioteca que consiste de un conjunto de esqueletos algorítmicos. Un esqueleto es una plantilla que incorpora los aspectos predeterminados de cada algoritmo para que el usuario del mismo, se preocupe únicamente de ingresar los datos particulares del problema que quiere resolver. Adicionalmente, MaLLBa, también permite al usuario programador, la solución de su problema utilizando computación distribuida, sin que el usuario tenga grandes preocupaciones con los detalles de comunicación y sincronización.

La biblioteca, implementada en C++, permite instanciar problemas de optimización combinatoria, utilizando esqueletos predefinidos, los cuales se corresponden con métodos de optimización, exactos, heurísticos, e híbridos. Luego, cada implementación de un método, se puede ejecutar secuencialmente, en paralelo utilizando una red de área local (LAN), o en una red de área extensa (WAN), por ejemplo Internet [LSCO].

Los problemas que se han resuelto, o que se podrían resolver utilizando MaLLBa, son en principio numerosos, en la Figura D.1.1 presentamos dos ejemplos.

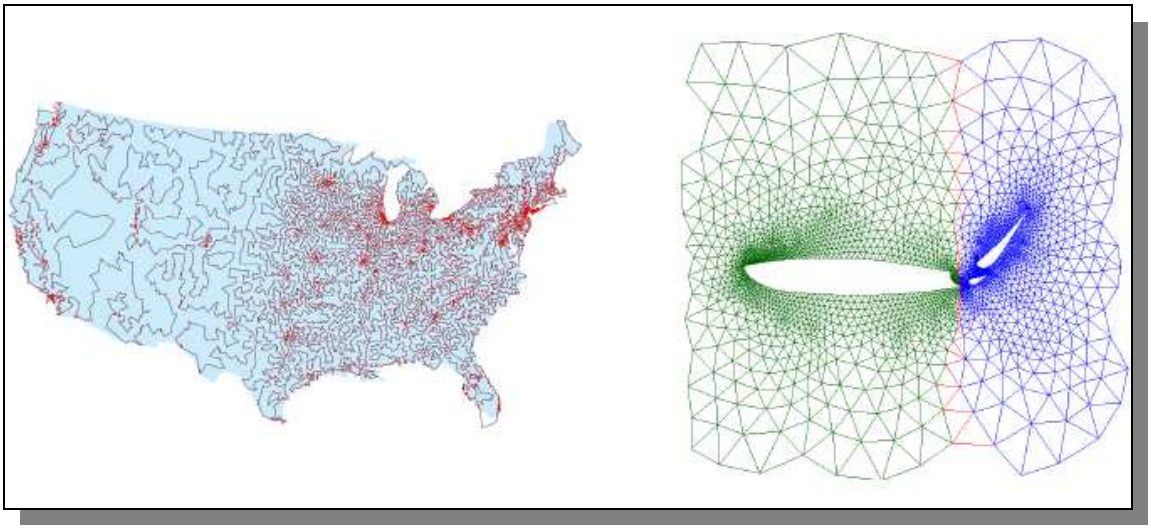

Figura D.1.1: El Problema TSP (Travel Salesman Problem) para EEUU y a la derecha la bisección del ala de un avión.

La metaheurística por la cual se utilizará la biblioteca MaLLBa, en este caso, será la de algoritmos genéticos. Sin embargo, MaLLBa implementa muchas más metaheurísticas, entre ellas: Simulated Annealing, CHC Method, Evolution Strategy, Ant Colony Optimization, GA+SA Hybrid Algorithm, Cooperative Local Search, y Particle Swarm Optimization[MALL].

D.2. Objetivos

El objetivo de MaLLBa es, simplificar la tarea del desarrollador que pretenda implementar un algoritmo utilizando una técnica en particular.

MaLLBa ofrece varias ventajas, entre ellas, permite al desarrollador implementar la solución escribiendo una cantidad mucho menor de líneas de código que los que se deberían implementar si se quisiera implementar todo el algoritmo desde cero. Por otro lado, MaLLBa, ofrece un diseño probado, modular, y conceptualmente muy claro.

D.3. Diseño

D.3.1. Consideraciones Generales

El diseño de MaLLBa, tal cual veremos más adelante, en el diagrama de clases de diseño, es modular y muy simple, enfocado en la idea de que la tarea de la persona que adapta el problema al esqueleto de MaLLBa sea lo más sencilla posible.

D.3.2. Arquitectura

La arquitectura de los esqueletos MaLLBa, consiste de dos partes bien diferenciadas, las clases provistas, que se distinguen con la palabra clave *provided*, y las clases requeridas, que a su vez, se distinguen con la palabra *required*.

Toda la parte genérica del algoritmo, está codificada en las clases provistas, y no es necesaria que sea modificada por el investigador que desee simplemente instanciar un problema. En el caso del desarrollo realizado para este proyecto, las clases provistas no se han modificado, solamente se han realizado algunas modificaciones en el esqueleto del algoritmo genético, para incluir la funcionalidad de coeficiente de mutación adaptativo.

Por otro lado, las características particulares del problema y la solución, están en las clases requeridas, cuyos métodos deben ser implementados íntegramente por el programador que desee instanciar cualquier problema [RIO04].

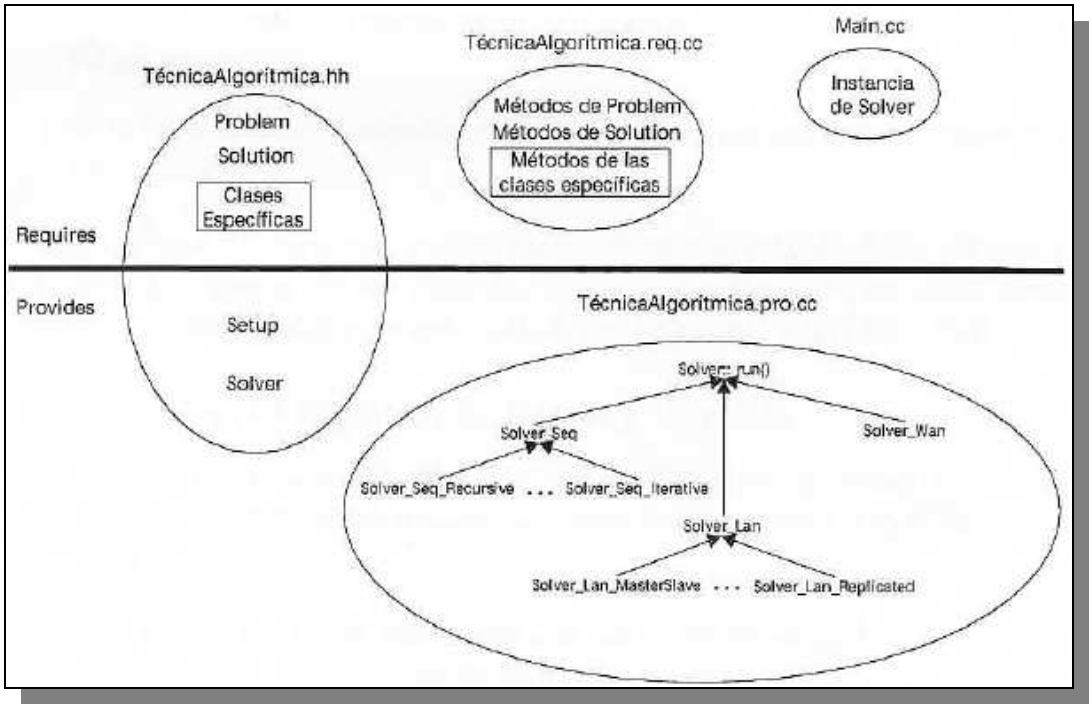

Figura D.3.2. Arquitectura de los esqueletos MaLLBa.

D.3.2.1. Tipos de usuario

Asociados a la arquitectura de los esqueletos y su uso e implementación, podemos encontrar diversos tipos de usuario de MaLLBa, los mismos se ilustran en la Figura D.3.2.1.1 [INST].

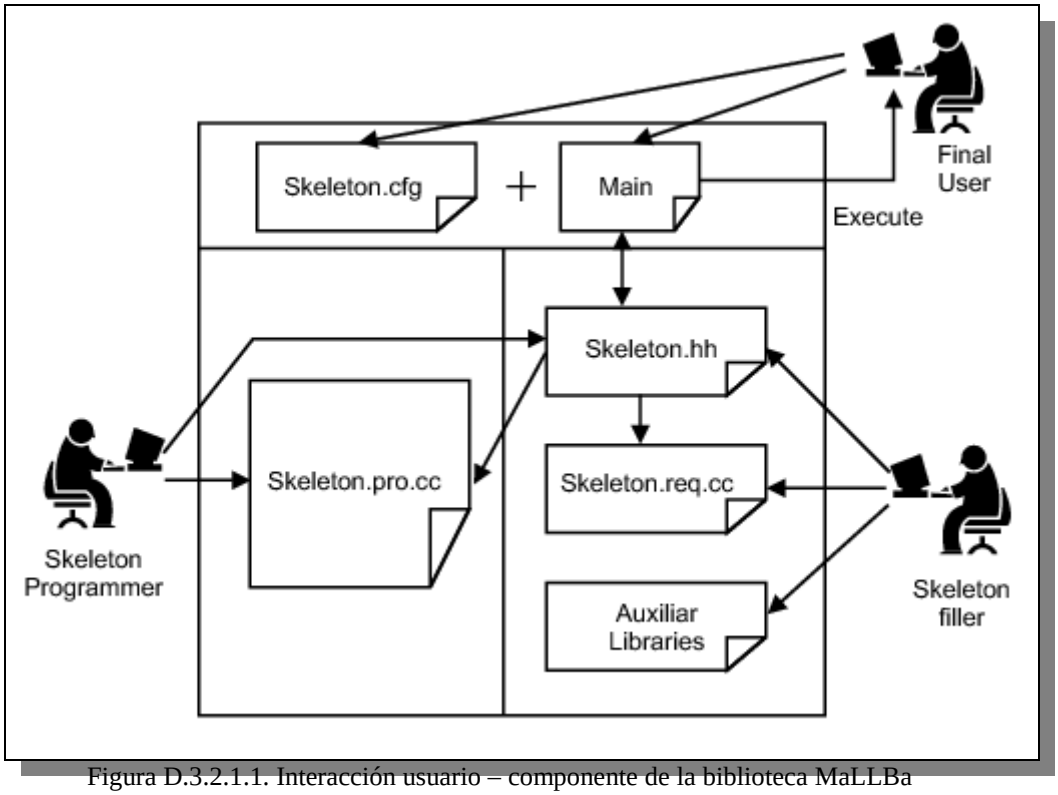

Figura D.3.2.1.1. Interacción usuario – componente de la biblioteca MaLLBa

Como podemos notar en la imagen, el llamado "Usuario Final", es el usuario que ejecuta un programa implementado en MaLLBa. El mismo, ejecuta un programa (*Main*) y modifica archivos de configuración para indicar parámetros de ejecución (cantidad de corridas independientes, por ejemplo).

El "*Skeleton Filler*" es un usuario programador que desea implementar la solución a un problema particular utilizando alguno de los esqueletos de MaLLBa, como tal, solamente implementa las clases identificadas como *required*.

Por su parte, el "*Skeleton Programmer*" es el usuario más avanzado de la biblioteca. Es un usuario que implementa esqueletos genéricos para que luego, sean utilizados por los *Skeleton Fillers*, es decir, genera o modifica las clases *provided*.

En el presente proyecto se ha trabajado en los tres niveles. Como usuarios, cada vez que se testeó el sistema como *Skeleton Fillers*, la mayor parte del tiempo, y como *Skeleton Programmers*, cuando modificamos el esqueleto de los algoritmos genéticos para que el mismo realice la adaptación dinámica de la probabilidad de mutación.

D.3.3. Diagrama de Clases de Diseño (DCD)

El diagrama de clases de diseño presentado en la Figura D.3.3.1, intenta presentar un vistazo general de las clases provistas y requeridas de MaLLBa.

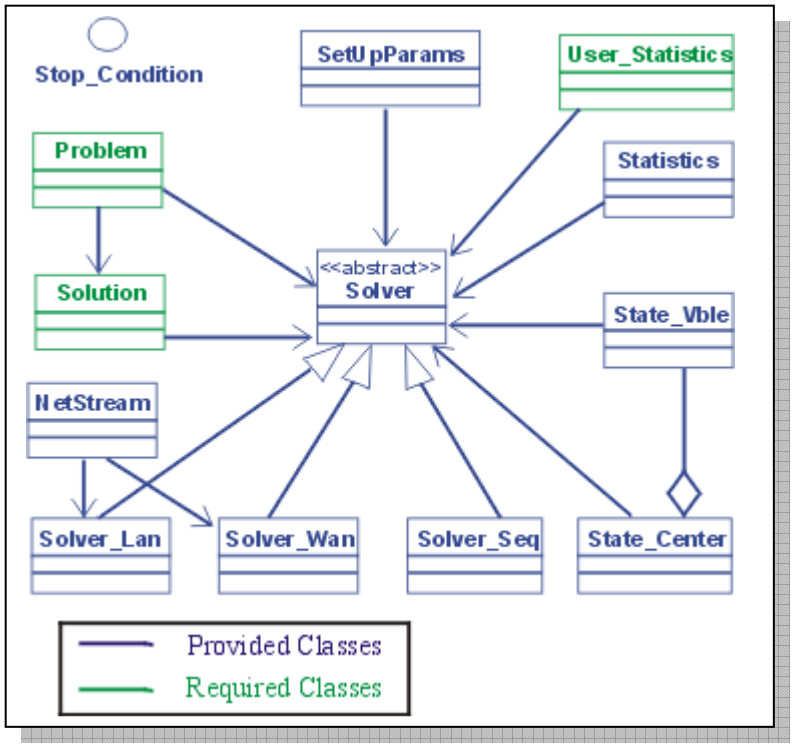

Figura D.3.3.1. Diagrama de clases de diseño de MaLLBa.

Lo primero que puede decirse del diagrama, es que tenemos tres subclases de la clase central, llamada *Solver*. Dichas subclases son, *Solver_Lan*, *Solver_Wan*, y *Solver_Seq*. Se podrá utilizar alguno de estos *solvers* en función de cuál sea el requerimiento de paralelismo. En el caso de este proyecto se utiliza el *solver* secuencial, ya que no se presentaron requerimientos de computación paralela. Queda como posible trabajo futuro la integración de dicha funcionalidad.

La clase *NetStream*, es utilizada para la comunicación entre procesos corriendo en distintos equipos, está construida sobre MPI[MPI] (*Message Passing Interface*), que es una especificación de interfaz para intercambio de mensajes. La implementación que se utiliza normalmente en Unix, y que el grupo de MaLLBa sugiere instalar, es MPICH[MPICH].

La clase *Statistics*, como su nombre indica, es la encargada de recolectar las estadísticas que se generan en la utilización de cada esqueleto algorítmico. La clase *User_Statistics*, permite almacenar estadísticas definidas por el usuario.

Las clases *StateVariable*, y *StateCenter*, son clases que mantienen el estado del algoritmo y permiten modificarlo. Para el caso del algoritmo genético, cuál es la iteración actual, cuáles son las probabilidades de mutación y cruzamiento, etc.

SetUpParams, es la clase que se encarga de mantener los datos de configuración del algoritmo. Por ejemplo: cantidad de ejecuciones independientes, tamaño de la población, etc.

Las clases requeridas *Problem* y *Solution* son las clases que debe codificar el *Skeleton Filler* para generar una implementación de un problema particular. Las mismas incluyen diversos métodos que son utilizados por el *solver* para la ejecución del algoritmo.

D.4. Relevamiento

Dado que el uso de MaLLBa no se impone como un requerimiento, se realizó un relevamiento de otras herramientas que proveen soporte a la implementación de algoritmos genéticos. En esta sección se muestra algunos de los aspectos principales de las opciones relevadas.

D.4.1. JGAP

JGAP (*Java Genetic Algorithms Package*)[JGAP] es una biblioteca para algoritmos genéticos desarrollada en *Java*, bajo la licencia GPL[GPL] y Mozilla[MOZ].

La biblioteca tiene documentación y claros tutoriales acerca de cómo comenzar a utilizarla. Sin embargo, es un proyecto nuevo, y no hemos encontrado referencias de utilización del mismo por parte del mundo académico.

D.4.2. GAlib

GAlib[GALIB] es una biblioteca de algoritmos genéticos orientada a objetos y desarrollada en C++. Al igual que MaLLBa, incluye características con el fin de realizar ejecuciones en paralelo, para lo cual, utiliza PVM[PVM] (*Parallel Virtual Machine*).

La biblioteca ofrece una documentación extensa que está cerrada desde el año 1996. Sin embargo, notamos que puede resultar complejo avanzar hacia una implementación paralela del algoritmo.

D.4.3. PGAPack

PGAPack[PGAP] (*Parallel Genetic Algorithm Library*) es una biblioteca desarrollada en C, que puede ser invocada, tanto desde código C, como Fortran. El diseño de la biblioteca, está realizado utilizando estructuras de datos que contemplan la filosofía de orientación a objetos.

La biblioteca permite, como su nombre indica, la ejecución en paralelo, y también permite hibridizar una heurística por parte del programador. Sin embargo, no resulta simple encontrar documentación específica acerca de su uso y de los resultados que se podrían esperar en su aplicación.

D.5. Conclusiones

Con tan solo realizar una rápida exploración de la web, es posible notar que la librería MaLLBa tiene gran apoyo en el ámbito académico, siendo, por lejos, la librería referenciada por más documentos y publicaciones - *papers* de instituciones educativas, tanto de España [PEASLP] (el país donde fue creada) como del resto del mundo[ITDP].

El diseño de MaLLBa, es modular y muy claro, y sigue el paradigma de orientación a objetos.

Si bien no hay gran cantidad de documentación acerca de su uso, en su web hay un tutorial que explica cómo instanciar un problema utilizando la técnica Taboo Search, pero además, para cada heurística disponible, la biblioteca viene con no menos de tres ejemplos listos para ejecutar.

Los ejemplos ejecutables, son muy claros, tanto que los consideramos ideales, para tener un punto de partida en la utilización de MaLLBa.

La biblioteca MaLLBa, disponible en su sitio web, tiene una distribución de las clases que nos resultó muy poco práctica para trabajar cooperativamente, el problema es que dicha distribución tiene todas las clases requeridas incluidas en un mismo archivo de extensión ".req.cc", lo que dificulta el trabajo en equipo, ya que casi siempre las modificaciones recaen sobre el mismo archivo, teniendo que realizar mucho trabajo en la sincronización (*merge*) de versiones. Lo que se hizo al respecto es rediseñar el código de manera de tener las clases requeridas distribuidas en archivos diferentes.

La biblioteca MaLLBa, si bien tiene una curva de aprendizaje alta en el primer momento, luego que se han sorteado las primeras dificultades, es una herramienta muy buena, flexible, y extensible.

D.6. Referencias

[OEGD] Optimización en Entornos Geográficamente Distribuidos. Proyecto MaLLBa. Enrique Alba y Carlos Cotta.

[MHCP93] Modern Heuristic Search Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford, C.R. Reeves (1993)

[LSCO] MaLLBa: A library of skeletons for combinatorial optimization. E. Alba et al.

[MALL] http://neo.lcc.uma.es/mallba/easy-mallba/html/algorithms.html

[RIO04] Revista de Investigación Operacional, Vol.25, No.1, I. Dorta, C. León, C. Rodriguez, I. Rojas (2004)

[INST] Skeleton Instantiation in the MaLLBa Library, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Catalunia, Barcelona, España.

[MPI] http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

[MPICH] http://www.mcs.anl.gov/research/projects/mpich2/

[JGAP] http://jgap.sourceforge.net/

[GPL] http://www.gnu.org/licenses/fdl.txt

[MOZ] http://www.mozilla.org/MPL/

[GALIB] http://lancet.mit.edu/ga/

[PVM] http://www.csm.ornl.gov/pvm/

[PGAP] http://www.aip.de/~ast/EvolCompFAQ/Q20_pgapack.htm

[PEASLP] Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance, *Information Processing Letters*, E. Alba (2001)

[ITDP] Integrating Task and Data Parallelism by means of Coordination Patterns. *The 6th International Workshop on High-level Parallel Programming Models and Supportive Environments* (HIPS'2001). LNCS vol. 2026, Springer-Verlag, pp. 16-27. San Francisco, USA, M. Díaz, B. Rubio, E. Soler, J.M Troya (2001)

E. Módulo de Extensión (EM)

E.1. Introducción

El módulo de reglas laborales permite declarar reglas de manera sencilla, las cuales luego se convertirán en restricciones del problema y también se encarga de la generación de turnos. A su vez, el módulo del algoritmo genético provee toda la lógica para lograr una optimización basada en algoritmos genéticos, no obstante puede suceder que se quiera optimizar alguna parte o algún algoritmo para que se realice cierta lógica que mejore los resultados, o se quiera realizar algún pre o post procesamiento de alguna solución obtenida. Para ello se cuenta con este módulo, que permite extender ciertos puntos para lograr mejores resultados en puntos en los cuales se determine que el algoritmo pueda ser mejorado.

Este documento incluye los fundamentos, detalles y aspectos técnicos de las estrategias, decisiones de diseño y funcionalidades implementadas en este módulo.

E.2. Objetivos y Requerimientos

E.2.1. Objetivos

El objetivo fundamental de este módulo es proveer la posibilidad de extender la algoritmia planteada, es decir, de poder crear funciones que permitan pre-procesar o post-procesar determinados puntos del algoritmo de optimización y de la generación de turnos, con el fin de obtener mejores resultados, soluciones más ajustadas a nuestras necesidades, sin tener que modificar el *core* de ambos módulos.

También se encarga de administrar todo el manejo que requieren las clases de extensión, al ser declaradas, dadas de altas, invocadas y destruidas.

E.2.2. Requerimientos

Los requerimientos de este módulo se resumen en:

- Definición, creación y destrucción de las clases de pre procesamiento.
- Definición, creación y destrucción de las clases de post procesamiento.
- Proveer a los otros módulos de las herramientas necesarias para poder obtener estas clases y utilizarlas con el fin de lograr la extensión.

E.3. Funcionalidades

E.3.1. Pre Procesadores

Esta funcionalidad, permite realizar algún procesamiento de la solución previo a la ejecución de cierta operación con el fin de obtener un mejor resultado.

Se han detectado 6 operaciones que se consideraron importantes de permitir realizar el pre procesamiento ya que son vitales para la generación de las soluciones, tanto dentro del algoritmo genético, como dentro de la generación de turnos. Las mismas son o forman parte de la inicialización, mutación, cruzamiento, generador de reglas de descanso, validador de reglas de relevo y generador de turnos.

E.3.1.1. PreExtensionCrossover

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en ambos individuos a cruzar, intentando obtener soluciones mejores antes de realizar el cruzamiento.

E.3.1.2. PreExtensionMutation

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución que se va a mutar intentando obtener soluciones mejores previo a la realización de la mutación.

E.3.1.3. PreExtensionInitialize

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución intentando obtener soluciones mejoradas previo a la búsqueda de soluciones iniciales.

E.3.1.4. PreExtensionBreakRuleGenerator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en el turno previamente a la asignación de los descansos, apoyándose en la solución que se tiene actualmente si es necesario.

E.3.1.5. PreExtensionReliefRuleValidator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en el turno previamente a la validación de los puntos de relevos de turnos, apoyándose en la solución que se tiene actualmente si es necesario.

E.3.1.6. PreExtensionShiftGenerator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución previamente a la generación de los turnos. Esta interfaz nos permite realizar algún tratamiento previo a la solución, ya sea de ordenamiento o de la índole que se quiera, logrando una mejora en la generación de turnos, pudiendo minimizar la cantidad de turnos o logrando una mejor distribución de los mismos.

E.3.2. Post Procesadores

Esta funcionalidad nos permite realizar algún procesamiento de la solución posterior a la ejecución de cierta operación con el fin de obtener un mejor resultado.

Al igual que para los preprocesadores, también hemos detectado 6 operaciones que se consideraron importantes de permitir realizar el post procesamiento ya que son vitales para la generación de las soluciones, tanto dentro del algoritmo genético, como en la generación de turnos. Las mismas son o forman parte de la inicialización, mutación, cruzamiento, generador de reglas de descanso, validador de reglas de relevo y generador de turnos.

E.3.2.1. PostExtensionCrossover

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en ambas soluciones que se cruzaron intentando obtener soluciones mejor preparadas para la generación de turnos, luego de realizar el cruzamiento.

E.3.2.2. PostExtensionMutation

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución luego de aplicarle la mutación.

E.3.2.3. PostExtensionInitialize

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución intentando obtener soluciones mejoradas, luego de la búsqueda de soluciones iniciales. Es importante de destacar que esta operación aplica para cada solución que se obtuvo y no para un conjunto de soluciones.

E.3.2.4. PostExtensionBreakRuleGenerator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en el turno luego de la asignación de los descansos, apoyándose en la solución que se tiene actualmente si es necesario.

E.3.2.5. PostExtensionReliefRuleValidator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución luego de la validación de los puntos de relevos de los turnos, apoyándose en la solución que se tiene actualmente si es necesario.

E.3.2.6. PostExtensionShiftGenerator

El objetivo de esta clase es proveer una interfaz cuyo fin sea el de buscar una mejora en la solución luego de la generación de los turnos. Esta interfaz nos permite realizar algún tratamiento posterior a la solución o al turno, ya sea de ordenamiento o de la índole que se quiera, logrando que la generación de turnos sea más eficiente, pudiendo reducir la cantidad de turnos o logrando una mejor distribución de los mismos.

E.3.3. ExtensionMgr

Esta clase tiene como objetivo principal controlar todo lo relacionado con las extensiones. Por lo tanto, una de sus principales funciones, es la de tener el control de cuáles son las clases que se han registrados para extender alguna funcionalidad, ya sea pre-procesadora o post-procesadora. Por otro lado, como se quiere realizar un manejo eficiente de memoria, al finalizar el algoritmo se elimina toda instancia creada, por lo tanto también se necesita almacenar el método que oficializará de destructor (por más información ver Anexo G). También nos va a proveer de las operaciones respectivas para obtener los pre o post procesadores que se necesiten.

E.3.4. Diseño

E.3.4.1. Consideraciones Generales

Como este módulo es enteramente de extensión, su diseño básicamente se compone por un conjunto de interfaces donde cada una representa una posible extensión, ya sea pre-procesadora o post-procesadora.

También cuenta con un *manager* que es el que se encarga de la creación, destrucción y de todo el manejo pertinente que se requiere para poder llevar a cabo la extensibilidad, es decir, mediante la ejecución de los métodos de cada una de ellas.

E.3.4.2. Diagrama de Clases de Diseño (DCD)

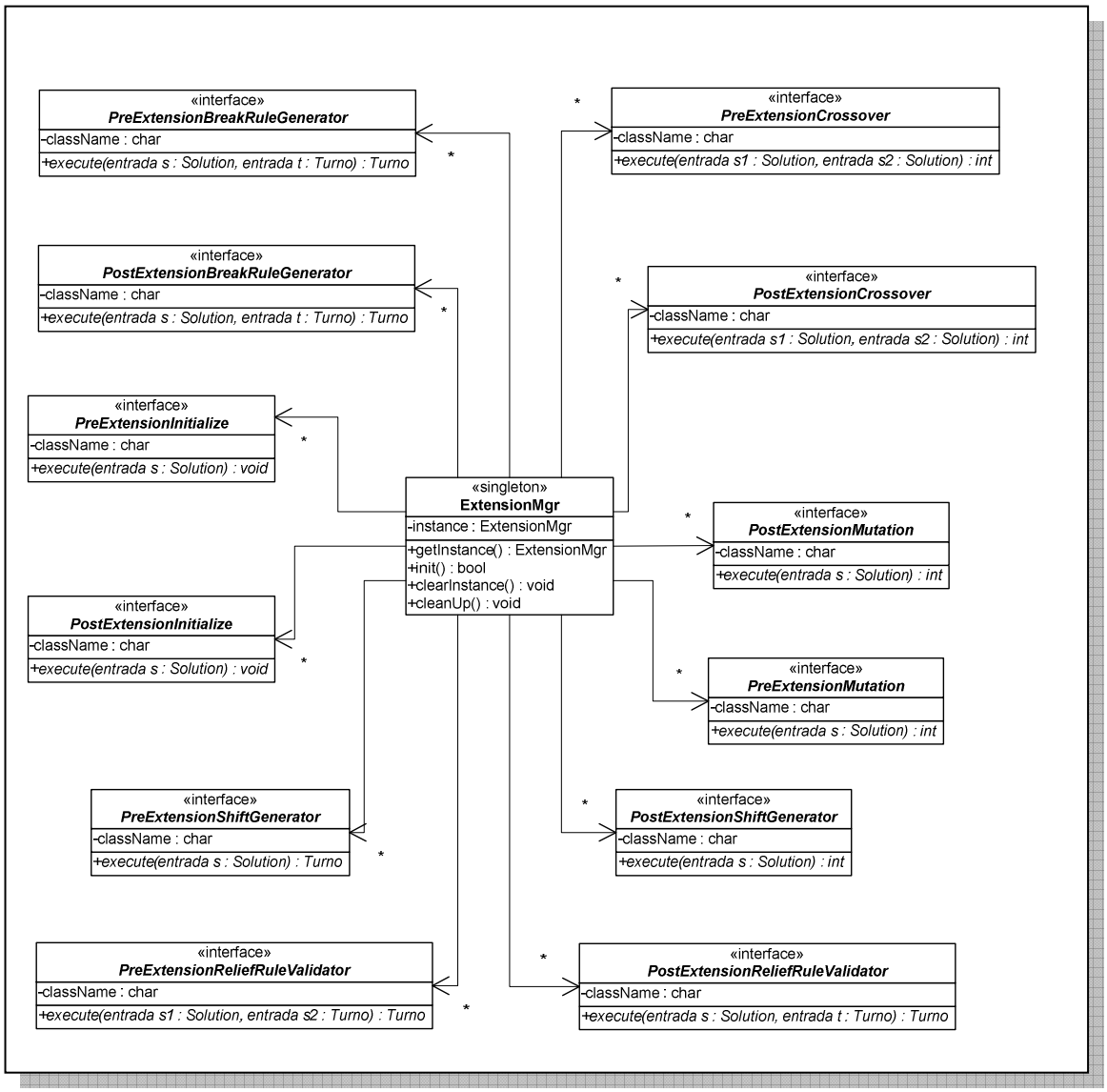

Fig. E.4.2.1 Diagrama de Clases de Diseño

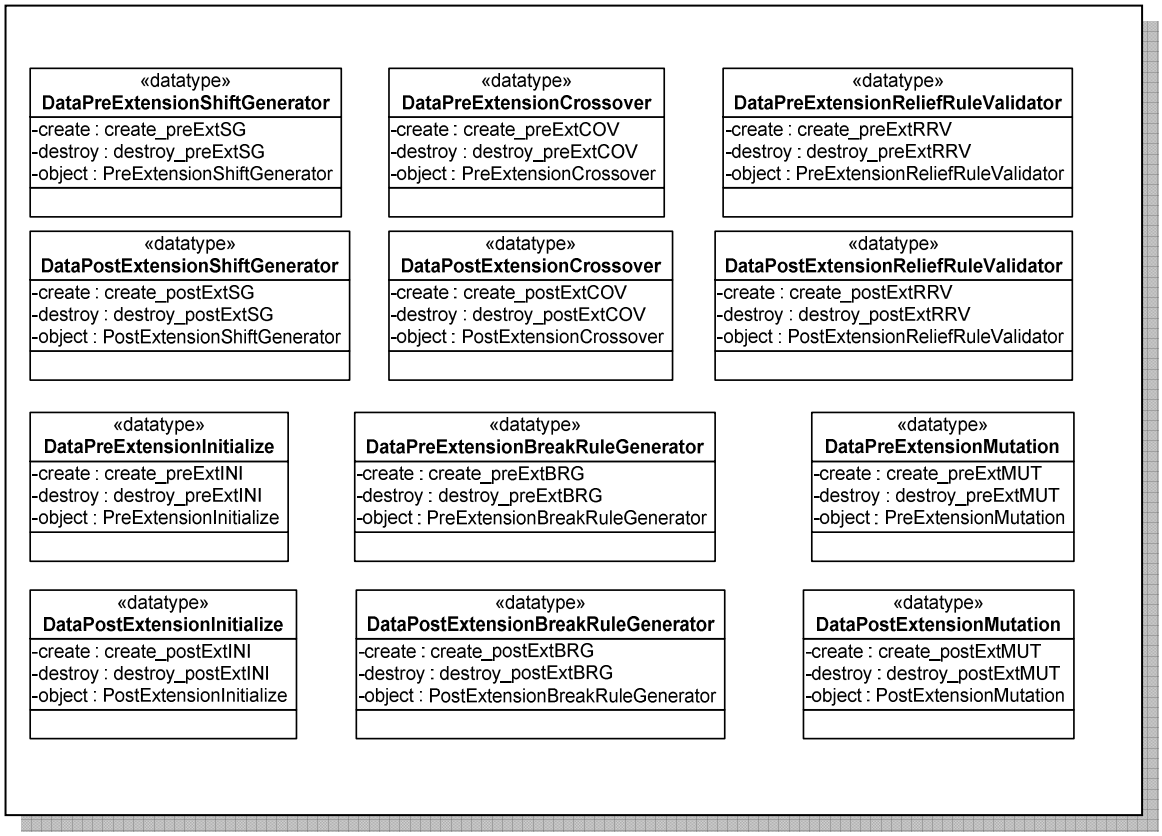

Fig. E.4.2.2 Continuación del Diagrama de Clases de Diseño

En la Figura E.4.2.2 se pueden observar la existencia de 12 *DataTypes* lo cuales son utilizados dentro del manejador (*ExtensionMgr*) para almacenar los datos de cada uno de los pre y post procesadores. Ellos contienen un atributo para poder instanciar el objeto que se desea, otro para destruirlo y el objeto en sí. Esto se debe a que luego de haberlo creado, debe ser invocado dentro de la lógica que corresponda para permitirnos lograr la extensibilidad que buscamos con el uso de este módulo.

Más adelante se verá en detalle cómo se definen los atributos para construir y destruir estos objetos.

E.4. Implementación

Para que funcione correctamente el módulo, es necesario que exista y esté definido con la correcta sintaxis el archivo de configuración. El mismo deberá llamarse *extension.xml* y debe ubicarse en la carpeta *conf*. Por más información ver Anexo G.

En la siguiente figura se presenta el pseudocódigo de la operación *init* que nos permite instanciar las clases, obteniendo la información del archivo correspondiente, y crear las estructuras necesarias para contar con las clases al momento de la ejecución de una extensión.

```
bool init()  {

      si (abrirArchivo(conf/extension.xml)  != ok)
            devolver false

      para cada pre y post procesador posible {
            si hay alguna clase definida en xml para ese pre o post
      {
                  abrir librería correspondiente
                  obtener constructor
                  obtener destructor
            }
      }

      para cada pre y post definido {
            instanciar la clase
            almacenar objeto en la estructura correspondiente
      }

      devolver true
}
```

Fig. E.5.1 Pseudocódigo de la operación *init*.

La Figura E.5.2, que se presenta a continuación, nos muestra el pseudocódigo de la operación *cleanUp* que se encargar de eliminar de la memoria todas las clases que forman parte del módulo así como de las estructuras creadas para contar con las clases al momento de la ejecución de una extensión.

```
void cleanUp()  {

      para cada pre y post definido {
            destruir la clase
            eliminar info del pre/post de las estructuras
      }

      cerrarLibrerias()
      destruirInfoLibrerias()
}
```

Fig. E.5.2 Pseudocódigo de la operación *cleanUp*.

F. Bibliotecas Auxiliares

F.1. Introducción

Para realizar determinadas funcionalidades se han utilizado bibliotecas que nos proveen la posibilidad de, al incluirla en el proyecto, utilizar las operaciones que nos brindan, con el fin de asegurarnos que existe cierto comportamiento que ya está implementado y bien definido, testeado y utilizado, lo que nos asegura que está funcionando, que la implementación es útil y con alta probabilidad de estar libre de errores importantes.

F.2. Objetivos

El objetivo de esta sección es proveer la información técnica de las bibliotecas que se han utilizado como complemento en el proyecto, explicando el fin para el cual se utilizó, cuáles son sus ventajas, desventajas, problemas que han surgido, y todo lo que resulte importante de destacar.

En la implementación, se han utilizado 2 bibliotecas auxiliares para complementar el proyecto. Una de ellas, y se podría considerar la más importante, es MaLLBa, que tiene un anexo particular en el cual se explica todos los detalles técnicos. (Véase anexo D).

La otra biblioteca que se utilizó se llama Mini-XML[MINI] y se utiliza para la lectura de los archivos de configuración. La versión utilizada es la 2.6 y puede ser descargada de la página oficial[DOWNLOAD].

F.3. Mini-XML

F.3.1. Introducción

Mini-XML es una pequeña biblioteca que se puede utilizar para leer y escribir archivos XML y archivos que contengan datos del estilo XML[XML] en donde una aplicación puede almacenar datos necesarios para la correcta ejecución de la misma.

F.3.2. Funcionalidades

Mini-XML soporta la lectura de archivos XML o strings con formato XML que estén escritos con encoding UTF-8 ó UTF-16.

Los datos son almacenados en una estructura de árbol con listas encadenadas, preservando la jerarquía de los datos respecto al formato XML, nombres, atributos. No existe límite en la cantidad de valores en los atributos, su tope es lo que soporte la memoria disponible.

F.3.3. Integración de la biblioteca

El objetivo principal de integrar esta biblioteca al proyecto, fue principalmente el de aprovechar la funcionalidad que nos brinda para la lectura de archivos XML.

Dado que estamos frente a un problema de optimización, es muy probable que se quieran hacer varias pruebas con distintos juegos de datos de entrada o variando ciertos valores que afectan el comportamiento del algoritmo o de la generación de los turnos. Es por este motivo que hemos tomado la decisión de contar con archivos de configuración que nos permitan obtener estos valores de forma que no se tenga que tocar el código, y a su vez, que el formato de los mismos sea XML, ya que es un formato universal y utilizado por muchas aplicaciones y frameworks como estándar para sus archivos de configuración.

Esta biblioteca nos brinda una interfaz muy práctica de utilizar la cual nos permite abrir, recorrer y escribir los archivos de manera simple. Las operaciones disponibles en la misma, pueden verse en el archivo *mxml.h*.

En la Figura F.3.3.1 se muestra de manera muy reducida alguna de las funciones que se presentan en la interfaz, algunas de ellas, han sido utilizadas en el proyecto.

```
extern void             mxmlAdd(mxml_node_t *parent, int where,
                        mxml_node_t *child, mxml_node_t *node);

extern void             mxmlDelete(mxml_node_t *node);

extern void             mxmlElementDeleteAttr(mxml_node_t *node,
                                    const char *name);

extern const char*      mxmlElementGetAttr(mxml_node_t *node,
                                          const char *name);

extern void             mxmlElementSetAttr(mxml_node_t *node,
                        const char *name, const char *value);

extern mxml_node_t      *mxmlFindElement(mxml_node_t *node,
                        mxml_node_t *top,
                        const char *name, const char *attr,
                        const char *value, int descend);

extern mxml_node_t      *mxmlLoadFile(mxml_node_t *top, FILE *fp,
                         mxml_type_t (*cb)(mxml_node_t *));

extern mxml_node_t      *mxmlLoadString(mxml_node_t *top,
                        const char *s,
                        mxml_type_t (*cb)(mxml_node_t *));

extern int              mxmlSaveFile(mxml_node_t *node, FILE *fp,
                                     mxml_save_cb_t cb);

extern int              mxmlSaveString(mxml_node_t *node,
                        char *buffer, int bufsize,
                        mxml_save_cb_t cb);
```

Fig. F.3.3.1 Algunas operaciones de la interfaz *mxml.h*

En la Figura F.3.3.1, podemos observar que podemos cargar a una estructura de datos un archivo XML simplemente pasándole un puntero al archivo e indicándole cual es el formato que contiene. También es muy sencillo obtener elementos, ya que simplemente invocando la función *mxmlFindElement*, pasándole el nodo a partir del cual queremos buscar y el nombre del elemento, nos devuelve el nodo de la estructura que estamos buscando.

Como se explicó anteriormente, esta biblioteca utiliza para el almacenamiento de datos, una estructura de árbol la cual es muy sencilla de interpretar. A continuación se muestra un ejemplo de un archivo XML y cómo queda su estructura de árbol.

```
<?xml version="1.0"?>
<data>
        <node>val1</node>
        <node>val2</node>
        <node>val3</node>
        <group>
            <node>val4</node>
            <node>val5</node>
            <node>val6</node>
        </group>
        <node>val7</node>
        <node>val8</node>
</data>
```

Fig. F.3.3.2 Archivo de muestra

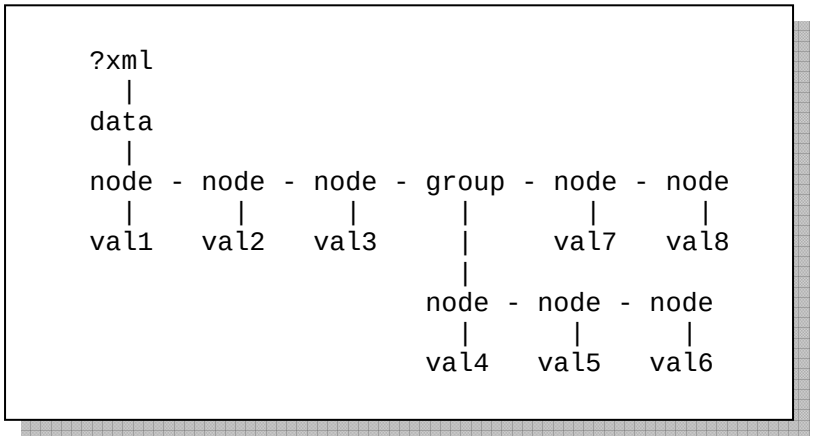

Fig. F.3.3.3 Estructura de árbol generada

En la Figura F.3.3.3 se observa la estructura del árbol, donde el "|" representa un puntero al primer nodo hijo y "-" representa el puntero al nodo siguiente o hermano. Es importante destacar, que el primer nodo que se tiene, es el de la declaración del archivo, de ahí que en la Figura F.3.3.3 vemos como nodo padre a *?xml*.

En la Figura F.3.3.4 se ve un ejemplo de código para cargar un archivo XML (*filename.xml*) usando la biblioteca.

```
FILE *fp;
mxml_node_t *tree;

fp = fopen("filename.xml", "r");
tree = mxmlLoadFile(NULL, fp, MXML_TEXT_CALLBACK);

fclose(fp);
```

Fig. F.3.3.4 Ejemplo de lectura

F.3.4. Ventajas y desventajas del uso de la biblioteca

Las ventajas que se han encontrado al utilizar esta biblioteca se podrían resumir en:

- facilidad para el manejo de archivos XML, ya sea carga, apertura, etc.
- muy útil para la búsqueda de elementos dentro del archivo.
- muy simple de integrar al proyecto para la codificación.
- instalación accesible.
- ocupa poco espacio en disco luego de instalada.
- muy buena documentación.

Si bien se ha encontrado positivo la integración de la biblioteca al proyecto, se podría mencionar como desventaja, que para obtener una nueva versión la cual contenga bugs solucionados, se debe volver a instalar la misma. Esto implica eliminar la versión anterior, realizar la instalación, verificación y pruebas de regresión correspondientes para asegurar su correcto funcionamiento.

F.3.5. Errores (*Bugs*)

En el proyecto, no se han encontrado *bugs* en la versión actual. Si se desean ver los *bugs* solucionados respecto a la versión anterior, se recomienda visitar la página oficial[BUGS].

F.4. Referencias

[MINI] Mini-XML - http://www.minixml.org/
[DOWNLOAD] Mini-XML download v2.6 - http://www.minixml.org/software.php?FILE=mxml/2.6/mxml-2.6.tar.gz&VERSION=2.6
[XML] XML - http://www.w3.org/XML/
[BUGS] Bugs Mini-XML - http://www.minixml.org/str.php

G. Instalación y Configuración

G.1. Instalación y Configuración de MaLLBa

La biblioteca MaLLBa, se obtiene en el sitio oficial[MaLLBa] de la misma. Para instalarla y utilizar su funcionalidad de ejecución paralela hay que tener instalado MPICH, una implementación de MPI (*Message Passing Interface*).

Para instalar MPICH en una distribución de la familia de Debian se puede ejecutar el comando:

```
> apt-get install mpich-bin
```

En otra distribución, hay que seguir las instrucciones del sitio de MPICH[MPICH].

Finalmente, MaLLBa, se instala en pocos pasos, en primer término, se descomprime el archivo .tar.gz:

```
> tar xzvf mallba.tar.gz
```

Luego, cambiamos al directorio principal:

```
> cd Mallba
```

A continuación, hay que configurar el entorno editando el archivo ***environment.***

Finalmente, se compila ejecutando:

```
> make all
```

Luego de ejecutados todos estos pasos, la carpeta de MaLLBa, queda como se muestra en la Figura G.1, resultando MaLLBa instalado.

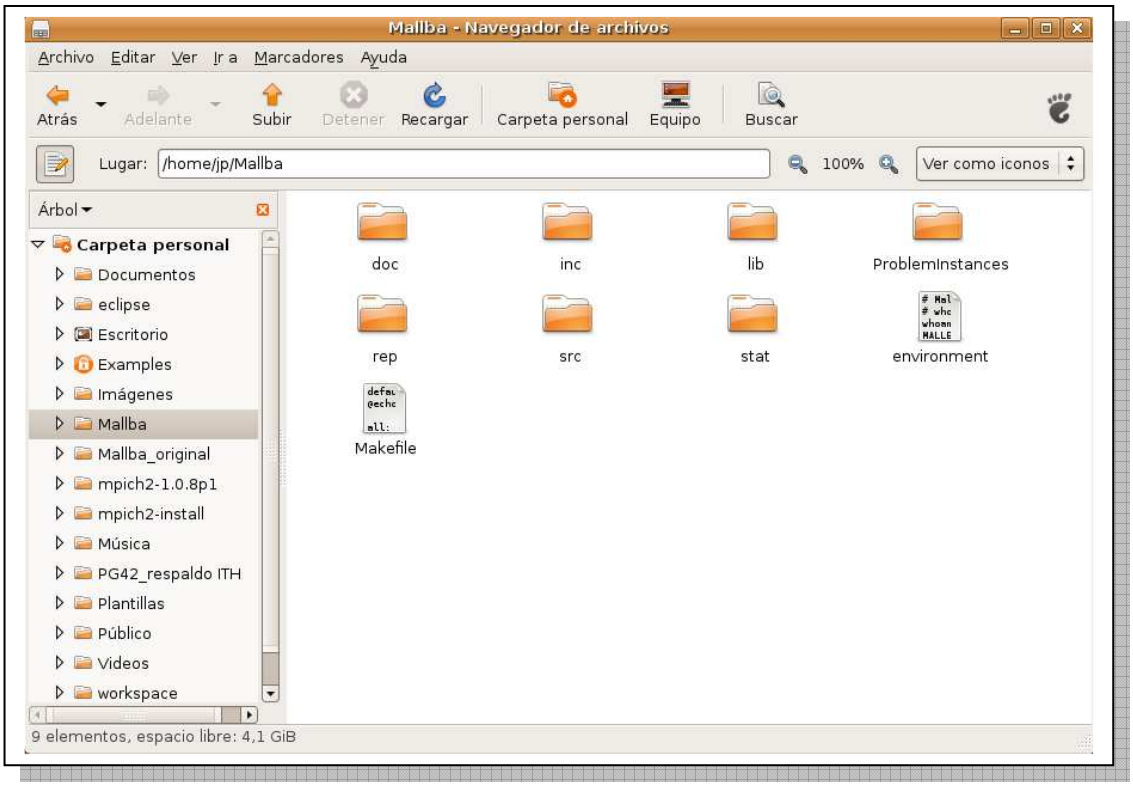

Fig. G.1 Carpeta MaLLBa

G.2. Instalación y Configuración de Mini-XML

Para obtener esta biblioteca basta con ir al sitio de Mini-XML[MINI] y descargar la versión 2.6[DOWN].

Luego se debe extraer el archivo, el cual genera una carpeta *mxml-2.6* con el contenido como se muestra en la Figura G.2.1.

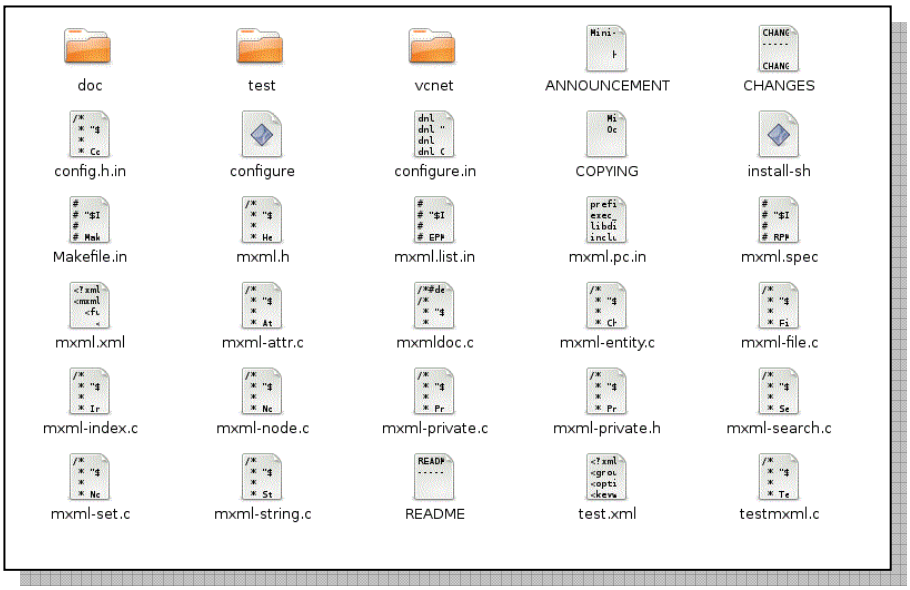

Fig. G.2.1 Carpeta *mxml-2.6*

Supongamos que se quiere instalar la biblioteca en el siguiente directorio:

- `/home/user/minixml`

Entonces lo que hay que hacer a continuación es, desde una consola, ingresar a la carpeta donde se extrajeron los archivos y ejecutar la siguiente secuencia de comandos:

- `./configure --prefix=/home/user/minixml`
- `make`
- `make install`

Esto generará la carpeta *minixml* en */home/user* con el contenido que se muestra en la Figura G.2.2.

Fig. G.2.2 Carpeta *mxml* instalada

Es importante verificar que en la carpeta *include* exista el archivo *mxml.h* y dentro de la carpeta *lib* se encuentre el archivo *libmxml.a*, como se muestra en la Figura G.2.3.

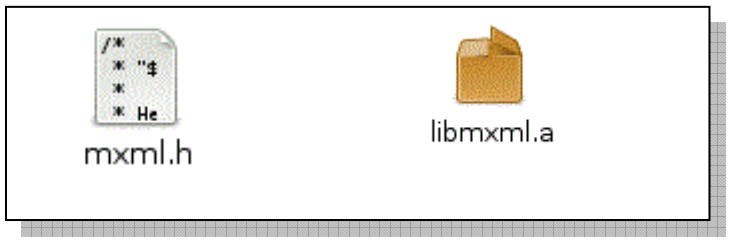

Fig. G.2.3 Verificación de archivos instalados

Es importante destacar que si se desea colocar esta carpeta en otra ruta, simplemente se debe cambiar la ruta al ejecutar el comando *configure* en el parámetro *prefix*.

La ubicación de la carpeta instalada se necesitará al momento de configurar la aplicación antes de ejecutarla. Mientras tanto, la carpeta en la que se extrajo no, por lo que se puede eliminar si se desea.

G.3. Despliegue de la Aplicación (*deploy*)

La carpeta PG42 contiene los fuentes necesarios para compilar y luego ejecutar la aplicación.

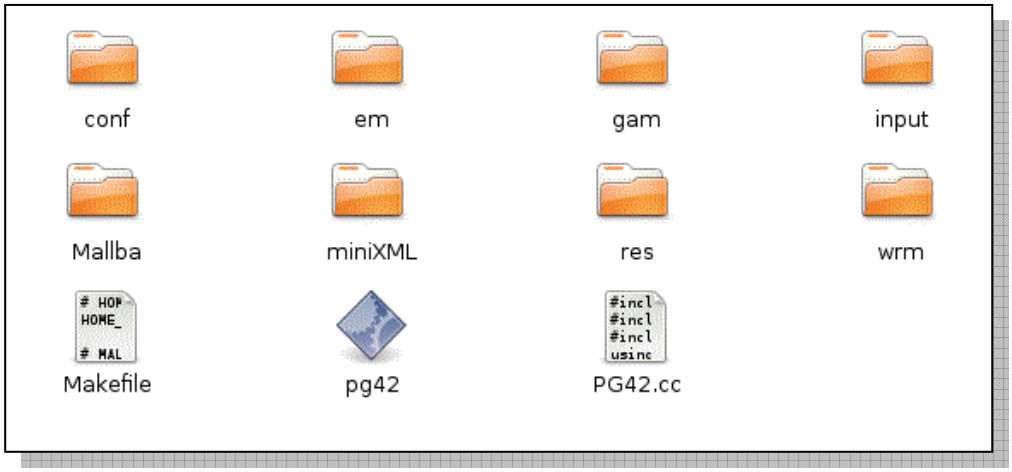

Fig. G.3.1 Componentes de PG42

La Figura G.3.1 muestra el contenido de la carpeta PG42, la cual contiene:

- 3 carpetas referidas especialmente a los módulos que componen el proyecto: El Módulo de Reglas Laborales (WRM), Módulo del Algoritmo Genético (GAM) y Módulo de Extensión (EM).
- carpeta *conf*: esta carpeta contiene todos los archivos de configuración para que el algoritmo funcione correctamente.
- carpeta *input*: en ella se colocarán los archivos de texto que tendrán la información particular del problema.
- carpeta *res*: aquí se guardará la solución obtenida
- *Makefile*: archivo para compilar y generar el ejecutable y bibliotecas
- *PG42.cc*: archivo *main* del programa.

También se puede apreciar en la Figura G.3.1 que existen las carpetas Mallba y miniXML, carpetas con el contenido de las bibliotecas que se utilizan, y además se muestra el ejecutable de la aplicación (*pg42*).

Las 3 carpetas referidas a los módulos, contienen solamente código fuente, ya sea archivos *headers* (.hh ó .h) y archivos con código fuente (.cc ó .c). Estos archivos no deberían ser modificados al momento del *deploy* de la aplicación.

G.3.1. Carpeta *conf*

La carpeta *conf*, contiene los archivos mostrados en la Figura G.3.1.1.

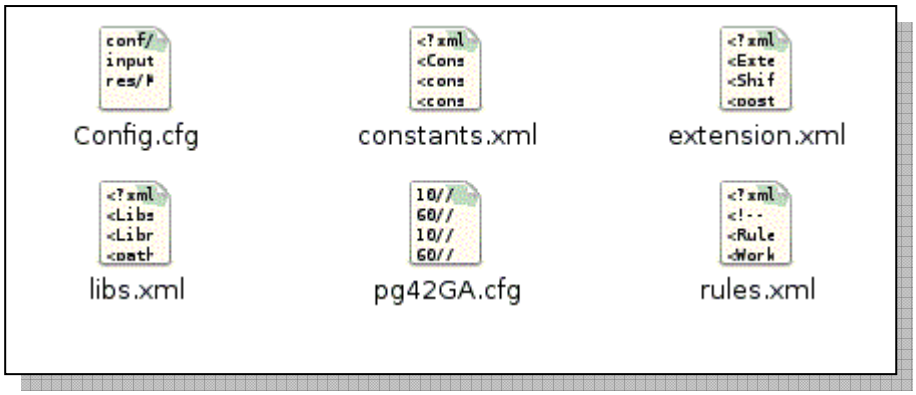

Fig. G.3.1.1 Contenido de la carpeta *conf*

Los archivos *constants.xml*, *extension.xml* y *rules.xml* son utilizados para la definición de constantes, extensiones y reglas laborales y descanso respectivamente. El contenido de los mismos será explicado a partir de la sección G.4.1. Lo mismo sucede con el archivo *libs.xml* que es usado para la creación de reglas laborales. Ahora solamente mencionamos que debe existir en esta carpeta pero su contenido será explicado más adelante.

El archivo *Config.cfg* contiene 3 líneas, donde la primera representa la ruta relativa al archivo que contiene los datos relativos del algoritmo genético. La segunda línea, representa la ruta relativa al archivo que tiene toda la información relativa a los viajes, depósitos, etc. Por último, la tercer línea, representa la ruta relativa al archivo que almacenará las estadísticas de las corridas.

En el archivo *pg42GA.cfg*, que se muestra en la Figura G.3.1.2, se puede observar que cada línea tiene un comentario con su contenido, él cuál es explicativo del mismo. La información que se determina en el archivo, corresponde a la cantidad de corridas independientes que se realizarán del algoritmo, la cantidad de generaciones que se evoluciona en cada corrida, la cantidad de individuos que hay en cada generación (el tamaño de la población), el operador de selección que se utilizará, las probabilidades de cruzamiento y mutación, y opciones de ejecución para correr el sistema en una red de área local.

```
1               // number of independent runs
1               // number of generations
100             // number of individuals
100             // size of offsprings in each
generation
1               // if replaces parents for
offsprings, or only offsprings may be new parents
0               // display state ?
Selections      // selections to apply
1 1             // selection of parents
1 0             // selection of offsprings
Intra-Operators// operators to apply in the
population
0 0.7           // crossover & its probability
1 1.0 0.1       // mutation & its probability
LAN-configuration
1               // refresh global state
0               // 0: running in asynchronized mode /
1: running in synchronized mode
1               // interval of generations to check
solutions from other populations
```

Fig. G.3.1.2 Contenido de la carpeta *conf*

G.3.2. Carpeta *input*

Dentro de la carpeta *input*, se encuentran los archivos en formato texto, que definen los valores de los datos de entrada para los distintos escenarios deseados. Por defecto se buscará cargar el archivo *DataEscenario.txt*, que deberá contener toda la información relativa a la cantidad de ciudades, distancias entre las mismas, cantidad de vehículos, depósitos y cantidad de vehículos en cada uno de ellos, tiempos que toma realizar los viajes expresos entre ciudades, cantidad de viajes y detalles de los mismos, información de puntos de relevos, etc.

El orden que debe tener esta información es la siguiente:

- cantidad de ciudades
- matriz de distancia entre las ciudades
- matriz de tiempos para viajes expresos
- cantidad de garages
- viajes, incluyendo:
    - ciudad origen
    - ciudad destino
    - hora salida
    - hora llegada
    - categorías habilitadas para realizar el viaje
- puntos de relevos

G.3.3. Makefile

El *makefile* tiene como objetivo, brindar reglas que permitan compilar el código y generar las bibliotecas básicas necesarias para una correcta ejecución de la aplicación.

Como primer punto importante a tomar en cuenta, es que se debe configurar en él, la carpeta base o raíz donde se ubicará la carpeta PG42. Este valor debe ser guardado en la variable HOME_DIR del *makefile*.

También se debe configurar la carpeta base o raíz donde se instaló la biblioteca *MaLLBa*, guardando este valor en la variable MALLBA_DIR.

Al igual que con *MaLLBa*, se debe configurar la carpeta base o raíz de la biblioteca *Mini-XML* y su valor se guardará en la variable MINIXML_DIR.

La última variable que se debe configurar es la ruta absoluta al directorio *bin* del *mpich*, responsable de la compilación y ejecución distribuida de la aplicación.

Además de estas variables, el *makefile* contiene todas las reglas para compilar los archivos de los 3 módulos, para la generación de las bibliotecas necesarias para la ejecución y creación del ejecutable de la aplicación.

G.4. Configuración de Reglas Laborales y Descanso

G.4.1. Reglas Laborales

Cuando se quiere agregar una nueva regla laboral, se deben modificar los archivos *rules.xml* y *libs.xml*.

Con respecto al primero de ellos, se debe agregar la regla como una nueva entrada, y la misma debe cumplir con el formato definido en la gramática. En la Figura G.4.1.1 se muestra un ejemplo.

```
<WorkerRules>
        <WorkerRule>OR(GREATER(SHIFTDURATION,"200"),
LESSER(MINSHIFTKMRELATION,"60"))</WorkerRule>
</WorkerRules>
```

Fig. G.4.1.1 Definición de una regla

En este ejemplo, podemos ver que se define una regla OR, donde cada uno de sus componentes hijos son comparadores. A su vez, cada componente hijo de ellos, es una condición de tipo parámetro. Algunas definidos por el usuario, como son

SHIFTDURATION y MINSHIFTRELATION, y otras son valores fijos como lo son 200 y 60.

Cuando se utiliza una condición, ya sea parámetro nuevo o alguna extensión de un comparador u operador lógico, debe especificarse dentro del archivo *libs.xml*, en qué biblioteca se encuentra la definición.

A continuación se muestra un ejemplo de la configuración de este archivo y se explica en detalle los valores que deben tomarse en cuenta al momento de definir una nueva biblioteca, con el fin de que quede correctamente definida.

```
<?xml version="1.0" encoding="utf-8"?>
<Libs>
    <Library type=cond>
        <path>wrm/Logical.so</path>
        <objects>
            <obj>AND</obj>
            <obj>OR</obj>
            <obj>NOT</obj>
        </objects>
    </Library>
    <Library type=cond>
        <path>wrm/Operator.so</path>
        <objects>
            <obj>LESSER</obj>
            <obj>GREATER</obj>
            <obj>EQUAL</obj>
        </objects>
    </Library>
    <Library type=cond>
        <path>wrm/Function.so</path>
        <objects>
            <obj>f_A</obj>
            <obj>f_B</obj>
            <obj>f_C</obj>
        </objects>
    </Library>
    <Library type=param>
        <path>wrm/Cond.so</path>
        <objects>
            <obj>SHIFTDURATION</obj>
            <obj>MINSHIFTKMRELATION</obj>
        </objects>
    </Library>
</Libs>
```

Fig. G.4.1.2 Ejemplo de *libs.xml*

Cada biblioteca nueva que se agregue, debe estar definida aquí como una nueva entrada bajo el tag *library*. Como valor del tipo de la biblioteca, hay que especificar si nos estamos refiriendo a una biblioteca que consta de la definición de parámetros o si es de operadores lógicos o de comparación. Para la primera categoría, basta con poner como valor del campo *type*, el string *param*, mientras que para el otro caso, se debería poner *cond*.

Además, debe especificarse la ruta relativa donde se ubicará la misma y una lista de los objetos que contiene y pueden ser utilizados dentro de la definición de reglas.

La siguiente figura muestra un ejemplo de cómo se implementa una clase que podría usarse como condición de una regla. Se mostrará la definición de ShiftDuration.

```
#include "Parameter.cc"
#include <stdio.h>

class ShiftDuration : public Parameter {

public:

    ShiftDuration () {
        sprintf (this->className, "%s", "SHIFTDURATION");
        this->values2return = 0;
    }

    virtual ~ShiftDuration() {}

    virtual int* evaluate (Solution* sol) {
        int quantity = sol->getCantidadTurnosLibroServicio();
        int* retVal = new int [quantity];
        for (int j=0 ; j < quantity; j++) {
            retVal[j] = sol->getLibroServicios()[j]->getDuracion();
        }
        this->values2return = quantity;
        return retVal;
    }
};

extern "C" Parameter* createSHIFTDURATION () {
    return new ShiftDuration ();
}

extern "C" void destroySHIFTDURATION (Parameter* param) {
    delete param;
}
```

Fig. G.4.1.3 ShiftDuration.cc

Como se aprecia en la Figura G.4.1.3, en el constructor se especifica el nombre con el cuál es identificado el objeto en el archivo *libs.xml* y será utilizado al momento de formar parte de una condición de una regla laboral.

Debe tener implementado el método *evaluate* con su correspondiente lógica y además tendrá que tener definidas las 2 operaciones para la creación y destrucción del objeto al cargar la biblioteca. Las mismas son las que están fuera de la definición de la clase y comienzan con la palabra *extern*. Se debe respetar el formato definido en la superclase, así como también el formato de los nombres de ellas. Es importante destacar

que estos nombres, deben definirse siempre de la siguiente manera: primero la palabra *create* y *destroy* seguido del identificador de la clase. Este identificador debe ser el mismo que se usa en el constructor de la clase.

G.4.2. Reglas de descanso

Cuando se desea agregar una nueva regla de descanso, basta con modificar el archivo *rules.xml*.

Dentro del objeto *BreakRules*, se debe agregar una nueva definición con los valores que se quieran para los 5 atributos. Es obligatorio que estén todos configurados para que la ejecución y evaluación de la misma sea satisfactoria.

A continuación se muestra un ejemplo de 2 reglas de descanso configuradas.

```
<BreakRules>
    <BreakRule>
        <duration>15</duration>
        <quantity>3</quantity>
        <start>60</start>
        <end>420</end>
        <gap>1</gap>
    </BreakRule>
    <BreakRule>
        <duration>30</duration>
        <quantity>1</quantity>
        <start>60</start>
        <end>420</end>
        <gap>1</gap>
    </BreakRule>
</BreakRules>
```

Fig. G.4.2 Definición de reglas de descanso

Los valores que se deben configurar para cada regla de descanso serán:

- duración del mismo (*duration*)
- cantidad de veces que puede ser asignado (*quantity*)
- a partir de qué minuto puede ser asignado, es decir, la cantidad de minutos que se ha de esperar, con respecto al inicio del turno, para poder realizar la primer asignación del mismo (*start*)

- hasta qué minuto puede ser asignado, es decir, la cantidad de minutos, con respecto al inicio del turno, hasta los cuales se puede realizar la última asignación del mismo (*end*)

- luego de cuántos minutos puede asignarse otro descanso de la misma regla (*gap*)

G.5. Configuración de Constantes

La aplicación permite al usuario declarar constantes, las cuales son identificadas por un nombre (*name*) y contienen un valor (*value*).

Estas constantes pueden ser utilizadas dentro de cualquier clase y lo que nos permiten es, poder realizar ejecuciones en paralelo de la aplicación, ya que las mismas se cargan al inicio de la aplicación. Simplemente se cambia el valor de las constantes que se quieran y se vuelve a ejecutar nuevamente sin afectar a las aplicaciones que se encuentran corriendo.

Otro punto importante a destacar es que no se necesita generar de nuevo el ejecutable, cosa que necesitaríamos si estas constantes fueran incluidas dentro de un archivo *header* (.h ó .hh).

Para obtener el valor de alguna de ellas, basta con obtener la instancia del controlador del módulo de reglas laborales (*WorkerRuleControllerImpl*) y ejecutar el método *getConstantValue*. Este nos devuelve un *string* con el valor de la constante. Esto nos permite que podamos obtener cualquier tipo primitivo, dejando para el usuario la correcta conversión al tipo deseado.

A continuación, se muestra como ejemplo el archivo de constantes, donde se muestra la definición de algunas constantes. Para definir una nueva constante basta agregar una nueva entrada con el nombre y valor deseado. En este caso, lo ejemplificamos en la constante de nombre *test3* y valor 40.

```
<?xml version="1.0" encoding="utf-8"?>
<Constants>
    <constant name=MAX_TIEMPO_TURNO value=480/>
    <constant name=MINTURNO value=300/>
    <constant name=test3 value=40/>
</Constants>
```

Fig. G.5.1 *Constants.xml*

G.6. Configuración de Extensiones

Si bien el algoritmo tiene un *core* de procesamiento, ya sea para el algoritmo genético como para el módulo de reglas laborales, el mismo nos permite extender algunas funcionalidades para lograr cierto comportamiento que sea particular.

Las clases que pueden extenderse son 6, ellas son:

- Crossover
- Initialize
- Mutation
- BreakRuleGenerator
- ReliefRuleValidator
- ShiftGenerator

Para cada una de ellas, existe un par de interfaces las cuales nos permiten hacer, o tener, un pre-procesamiento y post-procesamiento respectivamente. Las mismas son:

- PreExtensionCrossover
- PostExtensionCrossover

- PreExtensionInitialize
- PostExtensionInitialize

- PreExtensionMutation
- PostExtensionMutation

- PreExtensionBreakRuleGenerator
- PostExtensionBreakRuleGenerator

- PreExtensionReliefRuleValidator
- PostExtensionReliefRuleValidator

- PreExtensionShiftGenerator
- PostExtensionShiftGenerator

Cuando se desea extender alguna funcionalidad, simplemente se debe crear una clase que implemente la interfaz necesaria y se debe poner la lógica que se desea lograr en el método *execute*.

Es importante destacar que pueden existir toda la cantidad de extensiones que se quieran para una misma interfaz, basta agregarlas en el archivo *extension.xml*. El orden de ejecución es algo que hay que tener en cuenta al momento de declarar varias extensiones, ya que se ejecutarán en el mismo orden que fueron declaradas.

La Figura G.4.6.1 nos muestra la clase *ShiftQuantityOptimizer* la cual implementa la interfaz *PostExtensionShiftGenerator*.

```
#include "PostExtensionShiftGenerator.cc"
#include <stdio.h>
#include "shiftgenerator/ShiftGenerator.hh"

class ShiftQuantityOptimizer :public  PostExtensionShiftGenerator{
public:

    ShiftQuantityOptimizer(){
        sprintf (this->className, "%s", "SHIFTQUANTITYOPTIMIZER");
    }
    virtual ~ShiftQuantityOptimizer(){}

    virtual int execute (Solution* solution){
        // se implementa este método con la lógica que se quiera...
        return 0;
    }
};

extern "C" PostExtensionShiftGenerator* createSHIFTQUANTITYOPTIMIZER () {
    return new ShiftQuantityOptimizer ();
}

extern "C" void destroySHIFTQUANTITYOPTIMIZER (PostExtensionShiftGenerator* ext) {
    delete ext;
}
```

Fig. G.4.6.1 Ejemplo de extensión

Aquí podemos ver, al igual que al crear una condición, que se debe guardar en el atributo *className*, el valor con el cuál la clase será identificada dentro del archivo *extension.xml*. El mismo debe ir a continuación del nombre de los métodos *create* y *destroy*, que son usados por la biblioteca para crear y destruir los objetos.

En la siguiente figura vemos el archivo *extension.xml*, que es el encargado de contener todas las definiciones de las clases que representarán extensiones.

```
<?xml version="1.0" encoding="utf-8"?>
<Extension>
    <ShiftGenerator>
        <post>
            <libPath>em/ShiftGeneratorOptimizer.so</libPath>
            <objects>
                <obj>SHIFTQUANTITYOPTIMIZER</obj>
            </objects>
        </post>
    </ShiftGenerator>
</Extension>
```

Fig. G.4.6.2 Extension.xml

Para cada una de las clases principales que se pueden extender (*Crossover*, *Initialize*, *Mutation*, *BreakRuleGenerator*, *ReliefRuleValidator*, *ShiftGenerator*) existe su correspondiente definición de tag, con el mismo nombre, el cual nos permite declarar los pre y post procesadores que se utilizarán.

Bajo cada uno de esos tag, se puede especificar los pre y post con la correspondiente sección, como se ve en la Figura G.4.6.2. Para cada uno de ellos, se debe especificar el *path* relativo donde se encontrará la biblioteca que contiene la clase que hará la extensión, así como también el nombre de todos los objetos que serán invocados.

G.7. Referencias

[MaLLBa] MaLLBa - http://neo.lcc.uma.es/mallba/easy-mallba/index.html

[MPICH] MPICH, Implementation of MPI - http://www.mcs.anl.gov/research/projects/mpich2/

[MINI] MiniXML - http://www.minixml.org/

[DOWN] MiniXML source - http://www.minixml.org/software.php?VERSION=2.6&FILE=mxml/2.6/mxml-2.6.tar.gz