

1 Introducción

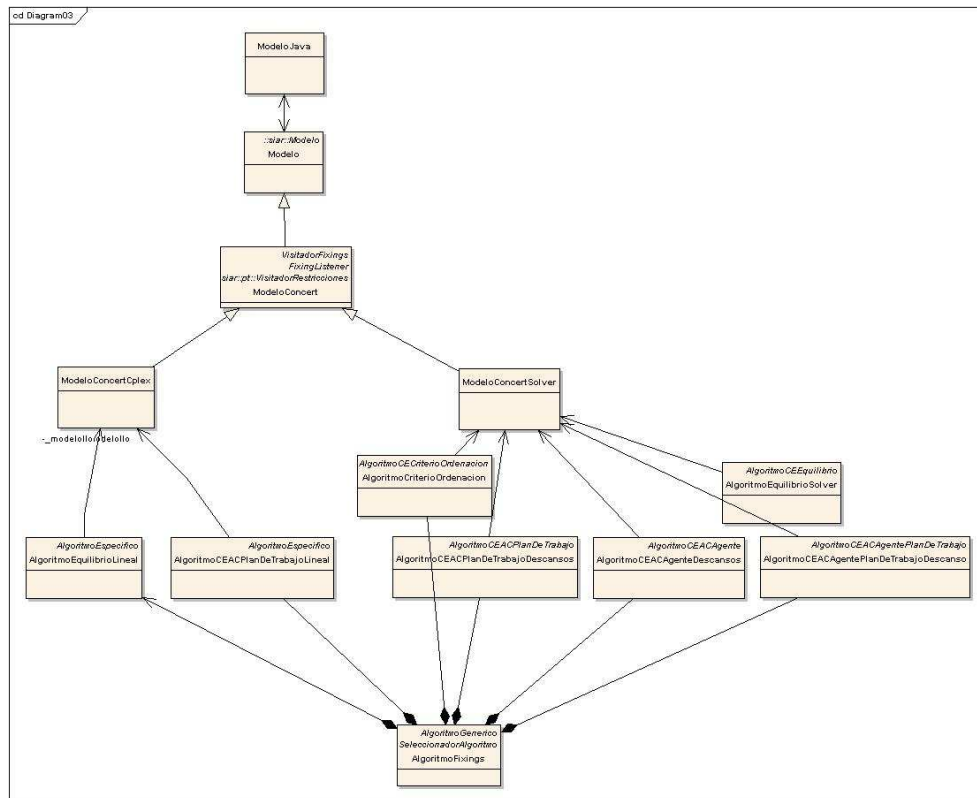
El objetivo de este documento es describir el proceso de optimización del Asignador de Planes de Trabajo, partiendo de la definición del Modelo de Asignación hasta la resolución final del problema y obtención de su solución.

1.1 Elementos presentes en el Proceso de Asignación

Durante el Proceso de Asignación, se van a utilizar los siguientes elementos:

- **Modelo de Asignación:** Es el punto de partida de la asignación, *y* representa en su totalidad el problema *a* resolver, desde un punto de vista de negocio (es decir, aparecen Agentes, Planes de Trabajo, Compatibilidades entre Agentes *y* Planes de Trabajo, etc.,)
- **Modelo de Optimización:** En función del Motor de Optimización que vaya *a* ser utilizado, será una formulación del problema contenido en el Modelo de Asignación en el formato definido por el Motor de Optimización desde el Modelo de Asignación:
 - Agentes de Asignación
 - Planes de Trabajo de Asignación
 - Restricciones
- **Algoritmo de Optimización:** Representa una técnica para resolver el problema definido por el Modelo de Asignación. Los algoritmos de Optimización solicitarán uno o varios de los Modelos de Optimización asociados a dicho Modelo de Asignación. En los Algoritmos de Optimización tendremos dos tipos de Algoritmos:
 - **Algoritmos Genéricos:** Son aquellos que son capaces (quizás mediante el uso de otros algoritmos) de resolver el problema para todos los Criterios de Evaluación
 - **Algoritmos Específicos:** Son aquellos que son capaces de resolver el problema pero teniendo en cuenta solamente un criterio de Evaluación. Estos Algoritmos serán utilizados por algunos de los Algoritmos Genéricos para resolver el problema.
- **Solución:** Es un conjunto de pares Agente – Plan de Trabajo, representando el hecho de que el Plan de Trabajo del par ha sido asignado al Agente. Diremos que una solución *S* es factible si cumple con todas las restricciones del Modelo de Asignación. Diremos que $(a, p) \in S$ si el Plan de Trabajo *p* ha sido asignado al Agente *a* en la Solución *S*.

Las relaciones entre estas entidades, así como los nombres de las Clases de Diseño, vienen descritos en el siguiente Diagrama de Clases.



Se puede observar que, en este diagrama, tenemos:

- Un Modelo de Asignación, representado por las clases *ModeloAsignacionPlanesDeTrabajo::Modelo* en Java y *siar::pt::Modelo* en C++.
- Dos Modelos de Optimización que heredan de la clase *siar::pt::ModeloConcert*:
 - *ModeloConcertCPLEX*: Representando un Modelo de Optimización basado en la tecnología ILOG CPLEX.
 - *ModeloConcertSolver*: Representando un Modelo de Optimización basado en la tecnología ILOG Solver.
- Siete Algoritmos Específicos:
 - *AlgoritmoCriterioOrdenacion*: Algoritmo basado en la tecnología ILOG Solver y utilizado para la resolución de Criterios de Evaluación por ordenación. Se puede observar que este Algoritmo hace uso del Modelo de Optimización *ModeloConcertSolver*.
 - *AlgoritmoCEACPlanDeTrabajoLineal*: Algoritmo basado en la tecnología ILOG CPLEX, y utilizado para la resolución de Criterios de Evaluación por atributo calculado. Se puede observar que este Algoritmo hace uso del Modelo de Optimización *ModeloConcertCPLEX*.
 - *AlgoritmoEquilibrioLineal*: Algoritmo basado en la tecnología ILOG CPLEX, y utilizado para la resolución de Criterios de Evaluación por equilibrio. Se puede observar que este Algoritmo hace uso también del Modelo de Optimización *ModeloConcertCPLEX*.
 - *AlgoritmoEquilibrioSolver*: Algoritmo basado en la tecnología ILOG Solver, y utilizado para la resolución de Criterios de Evaluación por equilibrio. Se puede observar que este Algoritmo hace uso también del Modelo de Optimización *ModeloConcertSolver*.

- AlgoritmoCEACAgenteDescansos: Algoritmo basado en la tecnología ILOG Solver, y utilizado para la resolución de Criterios de Evaluación por atributo calculado de agente. Se puede observar que este Algoritmo hace uso del Modelo de Optimización ModeloConcertSolver.
 - AlgoritmoCEACAgentePlanDeTrabajoDescansos: Algoritmo basado en la tecnología ILOG Solver, y utilizado para la resolución de Criterios de Evaluación por atributo calculado de agente-plan. Se puede observar que este Algoritmo hace uso del Modelo de Optimización ModeloConcertSolver.
 - AlgoritmoCEACPlanDeTrabajoDescansos: Algoritmo basado en la tecnología ILOG Solver, y utilizado para la resolución de Criterios de Evaluación por atributo calculado de planes. Se puede observar que este Algoritmo hace uso del Modelo de Optimización ModeloConcertSolver.
- Dos Algoritmos Genéricos:
 - *AlgoritmoFixings*: Utilizado para resolver los Modelos de Asignación cuando hay Criterios de Evaluación por Ordenación. Utiliza Algoritmos Específicos de todos los tipos.
 - *AlgoritmoLSDescanso*: Utilizado para intentar mejorar una primera solución encontrada con otros algoritmos. Está basado en ILOG Solver, en su funcionalidad de Búsqueda Local, y utiliza un Modelo de Optimización del tipo ModeloConcertSolver, y no utiliza otros algoritmos.

1.2 Descripción del Proceso de Asignación

En general, se podría resumir el Proceso de Asignación del siguiente modo:

- Se realiza un análisis del Modelo de Asignación, con objeto de decidir el o los algoritmos que serán utilizados para su resolución.
- Se ejecutan los algoritmos seleccionados.
- Se obtiene la solución final y se devuelve a Java

El proceso de asignación comienza con un Modelo de Asignación, creado mediante la ejecución del proceso de extracción, y representando por completo todos los aspectos del problema a resolver. A continuación se procederá a la resolución del modelo siguiendo estos pasos:

1. Selección del (los) algoritmos que serán utilizados para la solución del problema. Estos algoritmos serán genéricos, de modo que serán capaces de resolver el problema en su totalidad.
2. Ejecución de los algoritmos seleccionados. Eventualmente, alguno de estos algoritmos utilizará algoritmos específicos para resolver partes del problema
3. Obtención de la solución y devolución de la misma.

Como se puede observar, hay dos decisiones importantes a la hora de determinar el Algoritmo a utilizar:

- Elección del Algoritmo o secuencia de Algoritmos *Genéricos* que van a ser utilizados para resolver el Modelo de Asignación. Aquí tenemos dos posibles elecciones:
 - En el caso de que todos los Criterios de Evaluación sean por Coeficiente, encaja muy bien una definición Lineal del Modelo de Optimización. En este caso, utilizaremos un Algoritmo Genérico (*AlgoritmoFixings*) basado en algoritmos específicos lineales. La solución final obtenida será óptima, por lo que no hace falta utilizar ningún algoritmo de mejora de soluciones.

- En el caso de que exista algún Criterio de Evaluación por Ordenación, la definición lineal del modelo de Optimización se hace inestable, por lo que es necesario un Algoritmo Genérico(*AlgoritmoFixings*) que sea capaz de comunicar distintos tipos de Algoritmos Específicos y al ser posible que la solución provista por este Algoritmo no sea la óptima, a continuación se ejecutará un Algoritmo de Búsqueda Local con la intención de mejorar la solución obtenida (*AlgoritmoLSDescanso*).
- Elección del Algoritmo Específico dentro de cada uno de los Algoritmos Genéricos. Esta elección estará descrita en los apartados correspondientes a cada uno de estos Algoritmos Genéricos.

A continuación se procederá a describir cada uno de los componentes que forman parte del proceso de Asignación.

2 Modelo de Asignación (siar::pt:ModeloAsignacion)

El Modelo de Asignación es el punto de partida del Proceso de Optimización. Contendrá toda la información necesaria para definir cualquier Modelo de Optimización a resolver, esta información, descrita en los apartados posteriores, constará de:

- Agentes de Asignación
- Planes de Trabajo de Asignación
- Restricciones
- Criterios de Evaluación

El objetivo del Asignador será el de resolver este Modelo, encontrando una solución que:

- Asigne, a cada Agente de Asignación, o bien uno de los Planes de Asignación presentes en el Modelo de Asignación, o bien determine que el Agente queda NO _ASIGNADO.
- Cumpla todas las restricciones presentes en el Modelo de Asignación (descritas a continuación)
- Sea la mejor encontrada con respecto a los Criterios de Evaluación considerados de forma lexicográfica. En este sentido, si tenemos los Criterios de Evaluación C_1, \dots, C_n , y dos soluciones S_1, S_2 , se considerará mejor la Solución que sea *mejor* con respecto al Criterio de Evaluación que esté situado *antes* en la lista. Si, por ejemplo, la Solución S_1 fuese mejor con respecto a los Criterios C_2, \dots, C_n , pero peor con respecto al Criterio C_1 , se consideraría que la Solución S_2 es mejor que la Solución S_1 .

2.1 Agentes de Asignación (siar::Agente)

Son los agentes que, en función de las reglas aplicadas, son candidatos a ser asignados. Estos Agentes de Asignación podrían estar compuestos de dos Agentes físicos, de modo que la obtención de sus datos podría implicar a uno o a ambos agentes.

2.2 Planes de Trabajo de Asignación (siar::pt:PlanDeTrabajo)

Representan los Planes que van a ser asignados a los Agentes de Asignación. Cada uno de estos Planes de Asignación no tiene por qué corresponder, necesariamente, a un único Puesto de la Base de Datos. Podría perfectamente corresponder a varios Puestos de la Base de Datos, cubiertos durante periodos de tiempo disjuntos, o bien a puestos nuevos creados durante el proceso de extracción de los datos (normalmente planes de reserva).

Eso si, cada plan de trabajo tendrá un único grupo de descanso que será el que se siga durante todo el periodo de cobertura del plan.

Algunos Planes de Asignación vendrán marcados como Reserva Enumerada. Esto significará que estos Planes de Asignación serán cubiertos cuantas veces se desee, pero, en el caso de que el Criterio de Evaluación sea con respecto a coeficiente, cada vez que se cubran el coste de la asignación aumentará.

2.3 Restricciones (siar::Restriccion)

Las restricciones que se van a contemplar en el Modelo de Asignación son resultado de un filtrado realizado durante la ejecución de las reglas del proceso de extracción, así que no tienen por qué coincidir exactamente con elementos existentes en la Base de Datos. *Todas las restricciones existentes deberán cumplirse para que una solución sea válida.*

Por defecto, queda establecida la restricción de Unicidad de Agentes, que establece que a un Agente de Asignación sólo se le puede asignar un Plan de Trabajo de Asignación. Esta restricción no es configurable, y es de obligado cumplimiento sea cual sea el Modelo de Asignación a resolver.

Los Tipos de Restricción contemplados son:

- Compatibilidades.
- Cardinalidades.
- Restricciones de Relación entre Planes de Trabajo.
- Restricciones de Relación entre Agentes

2.3.1 Compatibilidades (siar::pt::Compatibilidad)

Expresan el hecho de que un Agente de Asignación sea compatible con un Plan de Trabajo de Asignación. En el caso de que el Agente de Asignación A no sea compatible con el Plan de Trabajo de Asignación P , entonces ninguna solución en la que el Agente de Asignación A sea asignado al Plan de Trabajo de Asignación P será válida.

En lo sucesivo diremos que $Compatible(A, P)$ será verdad si el Agente de Asignación A es compatible con el Plan de Trabajo de Asignación P con respecto a las Restricciones de Compatibilidad.

2.3.2 Cardinalidades (siar::pt::Cardinalidad)

Dado un Conjunto de Plan de Trabajos de Asignación $CONJ_P$, la restricción de Cardinalidad determina que los Plan de Trabajos de Asignación que pertenezcan a dicho conjunto pueden aparecer asignados un máximo de MAX veces en una solución para que esta última sea válida.

2.3.3 Restricciones de Relación entre Planes de Trabajos

Las restricciones de Relación entre Planes de Trabajo vienen definidas como un conjunto de tuplas(GD) de trabajos-grupos de descanso compatibles entre sí.

Cada tupla T_i informa de que las posibilidades de asignación de los trabajos-grupos de descanso a agentes ha de efectuarse según la información contenida en la tupla y en ese orden. Esto es, se establece que si tenemos una coleccion de tuplas de tamaño t :

Tupla 1 $\rightarrow \{ \dots, T_{1i}\text{-GD}_{1j}, \dots \}$

....

Tupla n $\rightarrow \{ \dots, T_{ni}\text{-GD}_{nj}, \dots \}$

las asignaciones a una t-upla de agentes del conjunto de agentes ha de efectuarse según la correspondiente i-tupla o bien no seguir ninguna de las tuplas.

Por ejemplo, si tenemos tres agentes de asignación y una serie de tuplas de tamaño 3..

Tupla 1 $\rightarrow \{T1-GD1, T2-GD3, T3-GD5\}$
Tupla 2 $\rightarrow \{T1-GD1, T2-GD5, T3-GD3\}$
Tupla 3 $\rightarrow \{T2-GD1, T3-GD5, T1-GD3\}$
Tupla 4 $\rightarrow \{T1-GD2, T2-GD4, T3-GD6\}$

si al primer Agente de Asignación se le asigna el trabajo-grupo de descanso T1-GD1 entonces a los otros dos agentes se les debe asignar las posibilidades T2-GD3, T3-GD5 o bien T2-GD5, T3-GD3. Ninguna otra posibilidad es valida, no puede ser asignado a ningún de los otros dos agentes por ejemplo los grupos de descanso de la Tupla 4 GD2, GD4 y GD6; por supuesto tampoco el GD1 y ni siquiera las combinaciones T3-GD5, T1-GD3.

Si al segundo Agente de Asignación se le asigna el trabajo-grupo de descanso T2-GD3 entonces no puede ser asignado al agente que queda, el grupo restante de la Tupla 2, T3-GD3.

Finalmente, el tercer agente debe quedar asignado al trabajo-grupo de descanso T3-GD5.

2.3.4 Restricciones de Relación entre Agentes (siar::pt:RelacionAgentes)

Las restricciones de Relación entre Agentes vienen definidas por una lista de Agentes de Asignación AG , una lista de Atributos Calculados ATR , y un conjunto t-uplas TP que asignan un valor a cada Agente de Asignación y Atributo Calculado contenido en los mismos. Vamos a determinar que $Valor(t, ac, ag)$ devuelve el valor para la t-upla t del Atributo Calculado ac y el Agente ag . Representando esto en forma de tabla:

	A_1			...	A_n		
	AC_1	...	AC_k		AC_1	...	AC_k
1	V_{111}	...	V_{1k1}	...	V_{11n}	...	V_{1kn}
...
m	V_{m11}	...	V_{mk1}	...	V_{m1n}	...	V_{mkn}

Para describir el comportamiento de esta restricción necesitaremos nomenclatura adicional. Si tenemos una Restricción de Relación entre Agentes ra , entonces:

- Llamaremos $ra.T(t, a)$ al conjunto de valores para la t-upla t y el Agente de Asignación a en la restricción de Relación entre Agentes ra , es decir, en nuestro caso, $\{V_{t1a}, \dots, V_{tka}\}$
- Si T es uno de los conjuntos de valores descritos con anterioridad, llamamos $P(T)$ al Conjunto de Planes de Trabajos de Asignación tales que los valores de los Atributos Calculados de la Restricción de Relación entre agentes computados sobre los mismos correspondan a los de T , es decir, $\{p \in P : AC_1.valor(p) = T_1, \dots, AC_k.valor(p) = T_k\}$
- Para cada Agente $a \in ra.AG$, CV_a^{ra} como el conjunto de conjuntos de valores definidos por la Restricción de Relación entre Agentes para el Agente a , Es decir, $CV_a^{ra} = \{ra.T(t, a) : t \in ra.TP\}$.
- Decimos ahora que, $\forall a_1, a_2 \in A, a_1 \neq a_2, \forall cv \in CV_{a_1}^{ra}, rel(cv, a_1, a_2) = \{ra.T(t, a_2) : t \in ra.TP \wedge ra.T(t, a_1) = cv\}$, es decir, $rel(cv, a_1, a_2)$ es el conjunto de conjuntos de valores que aparecen en la tabla para el Agente a_2 tales que la los correspondientes valores en la misma t-upla para el Agente a_1 son cv .

Dada esta nomenclatura, la Restricción de Relación entre Agentes establecería que, si un Agente A_s recibe un Plan de Trabajo de Asignación p tal que $p \in P(\text{ra}.T(t, A_s))$, entonces para todo $a \in \{A_1, \dots, A_n\} - A_s$, el Plan de Trabajo de Asignación asignado a a (p_a) deberá cumplir que $p_a \in \{P(\text{rel}(A_s, \text{ra}.T(t, A_s), a))\}$.

Vamos a ejemplificar estas definiciones, con objeto de clarificar los conceptos. Si tenemos una Restricción de Relación entre Agentes que determina que los Agentes a_1 y a_2 tienen que hacer turnos *distintos*, esto se representaría del siguiente modo (suponiendo que tenemos tres Turnos M, T, N , y tres Planes de Trabajos de Asignación P_M, P_T, P_N , con los Turnos definidos por sus subíndices):

	a_1	a_2
	TurnoPlan de Trabajo	TurnoPlan de Trabajo
1	M	T
2	M	N
3	T	M
4	T	N
5	N	M
6	N	T

Así, quedaría:

- Los Conjuntos de Valores ($\text{ra}.T(t, a)$) serían, para ambos Agentes, $\{M\}, \{T\}, \{N\}$
- Los Conjuntos de Plan de Trabajos por cada conjunto de Valores ($P(T)$):
 - $P(\{M\}) = \{P_M\}$
 - $P(\{T\}) = \{P_T\}$
 - $P(\{N\}) = \{P_N\}$
- Los Conjuntos de Conjuntos de Valores ($\text{CV}_{a_1}^{\text{ra}}$) serían, para ambos Agentes, $\{\{M\}, \{T\}, \{N\}\}$
- Ahora, los conjuntos relacionados ($\text{rel}(\text{cv}, a_1, a_2)$) serían:
 - $\text{rel}(\{M\}, a_1, a_2) = \{\{T\}, \{N\}\}$
 - $\text{rel}(\{T\}, a_1, a_2) = \{\{M\}, \{N\}\}$
 - $\text{rel}(\{N\}, a_1, a_2) = \{\{M\}, \{T\}\}$
 - $\text{rel}(\{M\}, a_2, a_1) = \{\{T\}, \{N\}\}$
 - $\text{rel}(\{T\}, a_2, a_1) = \{\{M\}, \{N\}\}$
 - $\text{rel}(\{N\}, a_2, a_1) = \{\{M\}, \{T\}\}$

Por lo tanto, la Restricción de Relación entre Agentes determinaría que Si al Agente a_1 le asignamos un Plan de Trabajo de Asignación con el turno $\{M\}$, entonces al Agente a_2 le tendremos que asignar un Plan de Trabajo de Asignación con uno de estos Turnos $\{\{T\}, \{N\}\}$, y así sucesivamente examinando todas las relaciones generadas.

2.4 Criterios de Evaluación (siar::pt::CriterioEvaluacion)

El Modelo de Asignación soporta dos tipos de Criterio de Evaluación en el caso de Asignaciones de Planes de Trabajos a Agentes:

- Criterios de Evaluación por Coeficientes: Asignan un coeficiente a cada Asociación de un Plan de Trabajo de Asignación con un Agente de Asignación. En función del objetivo que se espera del criterio se subdividen en:
 - Criterio de Evaluación por Coeficientes buscando el maximizar o minimizar una expresión en base a los coeficientes.

- Criterio de Evaluación por Coeficientes buscando equilibrar una expresión en base a los coeficientes obtenidos de un atributo calculado.
- Criterios de Evaluación por Ordenaciones: Se definen en función de listas de Preferencias y Prioridades

Ambos tipos quedan definidos a continuación.

2.4.1 Criterios de Evaluación por Coeficiente (siar::pt::CEAtributoCalculado o siar::pt::CEEquilibrio)

Estos Criterios de Evaluación proporcionan una serie de informaciones fundamentales:

- Por cada Agente de Asignación y cada Plan de Trabajo de Asignación, un coeficiente representando el *coste* de asignar dicho Agente de Asignación al Plan de Trabajo de Asignación. Diremos que si tenemos un Criterio de Evaluación por Coeficiente C^c , entonces $C^c.coef(a, p)$ será el coeficiente asociado a asignar al Agente a el Plan de Trabajo p .
- El *signo* de la evaluación, es decir, si queremos minimizar o maximizar los coeficientes obtenidos. El valor de $C^c.signo$ será -1 si queremos minimizar, y 1 si queremos maximizar.

Así, si tuviéramos una variable x_{ap} , valiendo 1 si el Agente a está asignado al Plan de Trabajo p y 0 en otro caso, un Criterio de Evaluación por Coeficiente(lineal) establecerá la siguiente función objetivo:

$$\{\min | \max\} \sum_{a \in A, p \in P} C^c.coef(a, p) x_{ap}.$$

mientras que un Criterio de Evaluación de Equilibrio(lineal) busca minimizar la varianza de suma de variables con el mismo coste, esto es:

$$\{\min\} \sum_{v \in V} \left(\sum_{AC.(a,p)=v} x_{ap} \right)^2$$

siendo $V = \{v \in C^c.coef(a, p) : a \in A, p \in PT\}$

Por razones de eficiencia y ahorro de memoria, se han definido tres subtipos de Criterios de Evaluación por Coeficiente, en función de la información que necesiten para calcular el mismo:

- *Basados en Agentes (siar::pt::CEACAgente)*: Sólo necesitan el Agente de Asignación para obtener el coeficiente. Dado un Agente de Asignación a , todas las llamadas a $C^c.coef(a, p)$ devolverán el mismo valor.
- *Basados en Plan de Trabajos (siar::pt::CEACPlanDeTrabajo)*: Sólo necesitan el Plan de Trabajo de Asignación para obtener el coeficiente. Dado un Plan de Trabajo de Asignación p , todas las llamadas a $C^c.coef(a, p)$ devolverán el mismo valor.
- *Basados en pares Agente-Plan de Trabajo (siar::pt::CEACAgentePlanDeTrabajo)*: Necesitan conocer tanto el Agente de Asignación como el Plan de Trabajo de Asignación para calcular el coeficiente.

Por otra parte, y afectando a este tipo de Criterios de Evaluación, aparece la *Reserva Enumerada* (RE). Esta Reserva Enumerada afecta en dos sentidos al Criterio de Evaluación:

- El Criterio de Evaluación debe determinar si tiene en cuenta o no la Reserva Enumerada para computar sus coeficientes
- En función de lo decidido en el punto anterior:
 - Si no se tiene en cuenta, para todo $p \in RE$, el *coste* de asignar p a un Agente a será siempre el mismo ($= C^c.coef(a, p)$).
 - Si se tiene en cuenta:
 - Se computará un *offset* del siguiente modo: $offset = (\max \{C^c.coef(a, p) : a \in A, p \in RE\} - \min \{C^c.coef(a, p) : a \in A, p \in RE\} + 1) \cdot (-C^c.signo)$.

- El *coste* asociado a las asignaciones de cada Plan de Trabajo $p \in RE$ vendrá representado por la siguiente función Lineal:

$$\left(\sum_{a \in A} C^c.coef(a, p) * asignado(a, p) \right) + offset * numAsig(p)$$

Siendo $asignado(a, p)$ igual a 1 si p está asignado a a en la solución, y 0 en otro caso, y $numAsig(p)$ igual al número de veces que p ha sido asignado a algún Agente.

Pero no todos los criterios de evaluación basados en coeficientes deben ser lineales; algunas veces, linealizar estos problemas resulta inconsistente (en memoria o por duración) y necesitamos usar Ilog Solver para conseguir ressolverlos.

Es por ello, que se presentan algoritmos no lineales para su resolución (que se explicarán con mayor detalle después) pero cuya heurística va encaminada por el mismo objetivo que lo explicado para cada uno de los criterios lineales explicados antes.

2.4.2 Criterios de Evaluación por Criterio de Ordenación (siar::pt::CECriterioOrdenacion)

Los Criterios de Evaluación por Criterio de Ordenación están formados por dos listas:

- Una lista de **Prioridades**, expresando la prioridad de los Agentes de Asignación para acceder a cada Plan de Trabajo de Asignación. Define, por cada Plan de Trabajo de Asignación, una Lista ordenada de Agentes de Asignación con respecto a su prioridad al acceder al Plan de Trabajo.
- Una lista de **Preferencias**, expresando la preferencia de los Agentes de Asignación por cada Plan de Trabajo de Asignación. Define, por cada Agente de Asignación, una lista ordenada de Plan de Trabajos de Asignación ordenada por preferencia del Agente.

Con esto, se definirá que una asignación (A_1, P_1) es conflictiva si existe otra asignación (A_2, P_2) tal que A_1 prefiere P_2 a P_1 (en función de la evaluación lexicográfica de CONJ_PREF), y además A_1 es más prioritario que A_2 para cubrir el Plan de Trabajo P_2 . Diremos que un Conflicto((A_1, P_1) , (A_2, P_2)) será VERDAD si se produce esta situación. El objetivo de estos Criterios de Evaluación será el de minimizar el número de estos conflictos.

Como hemos dicho anteriormente, un Criterio de Evaluación por Ordenación viene definido, fundamentalmente, por dos Entidades: **Preferencias** y **Prioridades**. A continuación se describe cada una de ellas:

- **Preferencias**: Una preferencia es una clase que representa la *preferencia* de un Agente de Asignación sobre Planes de Trabajos de Asignación con respecto a un aspecto concreto (Turno, Distancia a Localización de Entrada, etc.,).

En general una Preferencia proporciona un valor para cada Plan de Trabajo, y una definición de la Preferencia del Agente en función de dichos valores. Para ello, debe implementar una característica de cada una de las siguientes perspectivas:

- *Información Necesaria para obtener el Valor*: Con respecto a esta perspectiva, tenemos las siguientes posibilidades:
 - *Simple*: Sólo necesita conocer el Plan de Trabajo de Asignación para conocer el valor (e.g. Preferencia por Turno).
 - *Doble*: Necesita conocer tanto el Plan de Trabajo de Asignación como el Agente de Asignación para conocer el Valor (e.g. Preferencia por cercanía a Localización de Entrada)
- *Condicionalidad*: Si la expresión de la preferencia depende de una condición evaluada sobre el Plan de Trabajo de Asignación. Aquí tenemos tres posibilidades:
 - *No Condicionada*: La expresión de la preferencia no depende de ninguna condición evaluada sobre el Plan de Trabajo (e.g. Preferencia por cercanía a Localización de Entrada)

- *Condicionada Simple*: La expresión de la preferencia depende de una condición evaluada sobre el Plan de Trabajo, y para calcular dicha condición sólo se necesita conocer el Plan de Trabajo
- *Condicionada Doble*: La expresión de la preferencia depende de una condición evaluada sobre el Plan de Trabajo, y para calcular dicha condición se necesita conocer tanto el Plan de Trabajo como el Agente de Asignación
- *Forma de Expresar la Preferencia*: Existen dos formas de expresar la preferencia:
 - *Enumerada*: Se expresa mediante una lista ordenada, para cada Agente, de los valores asociados a los Planes de Trabajos de Asignación. Los Valores que vayan antes en dicha lista serán los preferidos por el Agente (e.g. Preferencia por Petición de Turnos).
 - *Inferida*: Se expresa mediante un orden sobre los valores obtenidos de los Plan de Trabajos (e.g. Cercanía a Estación de Entrada). Este orden puede ser uno de estos tres:
 - ORDEN_CRECIENTE: Se prefieren los valores en orden creciente
 - ORDEN DECRECIENTE: Se prefieren los valores en orden decreciente
 - ORDEN_INDIFERENTE: Para el Agente no es importante esta preferencia
- **Prioridades**: Una prioridad es una clase que representa la *prioridad* de los Agentes de Asignación para acceder a un Plan de Trabajo de Asignación con respecto a un aspecto concreto (Escala-fón, Tipo de Agente, etc.,).
 En general una Prioridad proporciona un valor para cada Agente, y una definición de la Prioridad de los Agentes con respecto al Plan de Trabajo en función de dichos valores. Para ello, debe implementar una característica de cada una de las siguientes perspectivas:
 - *Información Necesaria para obtener el Valor*: Con respecto a esta perspectiva, tenemos las siguientes posibilidades:
 - *Simple*: Sólo necesita conocer el Agente de Asignación para conocer el valor (e.g. Prioridad por Escala-fón).
 - *Doble*: Necesita conocer tanto el Plan de Trabajo de Asignación como el Agente de Asignación para conocer el Valor (e.g. Prioridad por Condicionante de Vacante)
 - *Condicionabilidad*: Si la expresión de la preferencia depende de una condición evaluada sobre el Agente de Asignación. Aquí tenemos tres posibilidades:
 - *No Condicionada*: La expresión de la prioridad no depende de ninguna condición evaluada sobre el Agente (e.g. Prioridad por Escala-fón)
 - *Condicionada Simple*: La expresión de la prioridad depende de una condición evaluada sobre el Agente, y para calcular dicha condición sólo se necesita conocer el Agente
 - *Condicionada Doble*: La expresión de la prioridad depende de una condición evaluada sobre el Agente, y para calcular dicha condición se necesita conocer tanto el Plan de Trabajo como el Agente de Asignación
 - *Forma de Expresar la Prioridad*: Existen dos formas de expresar la preferencia:
 - *Explícita*: Cuando para un mismo Plan de Trabajo, dos agentes siempre tienen el mismo orden relativo, es decir, cuando, para cada Plan de Trabajo, se puede extraer una lista de los agentes ordenados por su prioridad con respecto a este Plan de Trabajo
 - *Enumerada*: Se expresa mediante una lista ordenada, para cada Plan de Trabajo de Asignación, de los valores asociados a los Agentes de Asignación. Los Valores que vayan antes en dicha lista serán los prioritarios para el Plan de Trabajo.

- *Inferida*: Se expresa mediante un orden sobre los valores obtenidos de los Agentes (e.g. Prioridad por Escalafón). Este orden puede ser uno de estos tres:

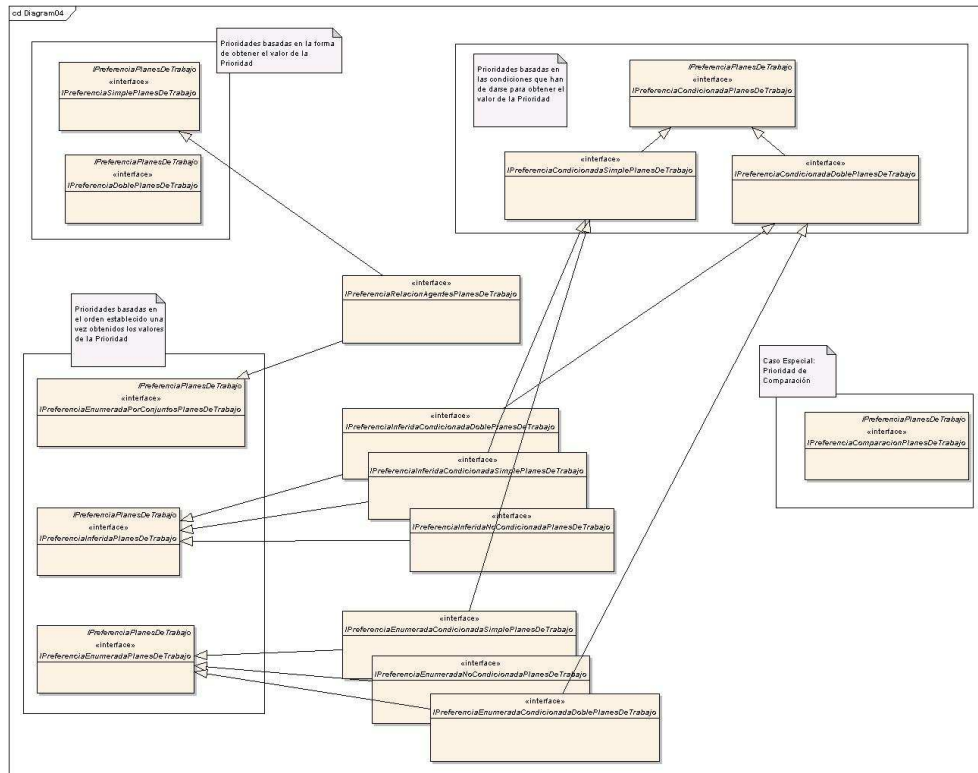
- ORDEN_CRECIENTE: Son prioritarios los valores en orden creciente
- ORDEN DECRECIENTE: Son prioritarios los valores en orden decreciente
- ORDEN_INDIFERENTE: Para el Plan de Trabajo de Asignación no es importante esta prioridad

- *Implicada*: Cuando para un mismo Plan de Trabajo, dos Agentes tienen un orden relativo distinto dependiendo de las asignaciones que se les hubiera realizado a los mismos. Así, si el Agente A_1 tiene asignado el Plan de Trabajo P_1 , entonces es más prioritario para el puesto P_2 que el agente A_2 . Ahora bien, si el Agente A_1 hubiese tenido asignado el Plan de Trabajo P_3 , entonces no sería más prioritario.

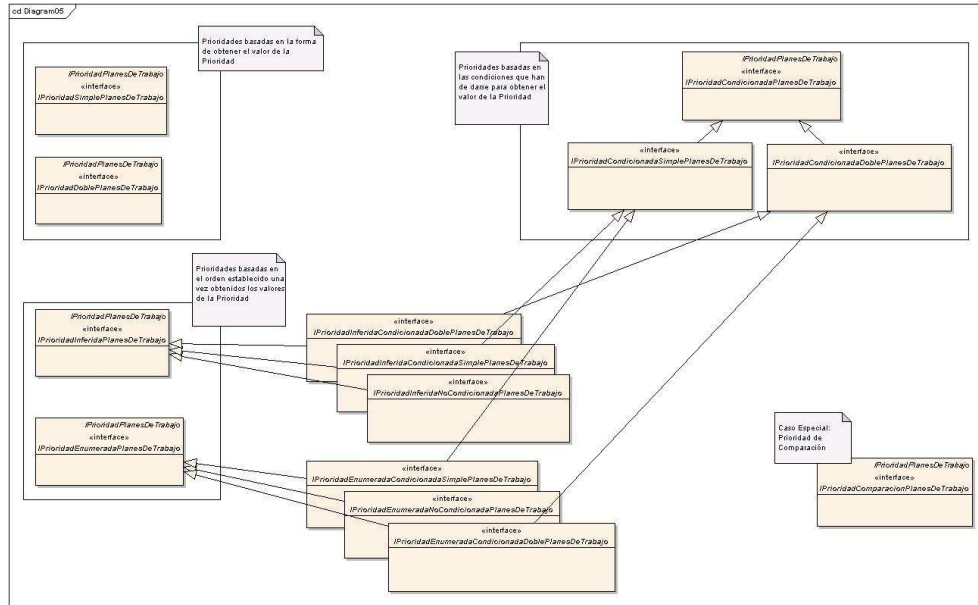
En la actualidad, en las Asignaciones de Planes de Trabajos, esta prioridad se manifiesta en la prioridad de agentes por baremo de descanso de SD.

Así, tendremos las siguientes Clases para representar las preferencias y prioridades existentes:

- **Preferencias:**



- **Prioridades:**



3 Modelos de Optimización

Los Modelos de Optimización son las representaciones en un formato comprensible por los Motores de Optimización disponibles del Modelo de Asignación a resolver. En la actualidad, usamos tres Motores de Optimización que necesitan de dos Modelos de Optimización distintos:

- Motor de Optimización CPLEX: Necesita de un Modelo de Optimización específico CPLEX (siar::pt::lineal::ModeloConcertCPLEX)
- Motor de Optimización Solver: Necesita de un Modelo de Optimización específico Solver (siar::pt::ModeloConcertSolver).
- Motor de Optimización de Búsqueda Local: Necesita de un Modelo de Optimización específico Solver (siar::pt::ModeloConcertSolver).

A continuación se describen los distintos Modelos de Optimización necesarios, cada uno de ellos en la forma más apropiada en función de su composición.

3.1 Descripción de los elementos contenidos en el Problema

En el Modelo de Optimización tendremos:

- *A*: Representando al conjunto de Agentes de Asignación del Modelo de Asignación
- *P*: Representando al conjunto de Plan de Trabajos de Asignación del Modelo de Asignación
- *T*: Representando al conjunto de Trabajos del Modelo de Asignación
- *D*: Representando al conjunto de Descansos del Modelo de Asignación
- *C*: Representando al conjunto de Restricciones de Compatibilidad contenidas en el Modelo de Asignación
- CP: Representando al conjunto de Cardinalidades Puras contenidas en el Modelo de Asignación
- RPT: Representando al conjunto de Relaciones entre planes contenidas en el Modelo de Asignación

- R : Representando al conjunto de Restricciones de Relación entre Agentes contenidas en el Modelo de Asignación

3.2 Modelo CPLEX (siar::pt::ModeloCPLEXCPX)

Para la representación de este Modelo de Optimización utilizaremos su formulación matemática, ya que es la más apropiada para este tipo de Modelos.

Dividiremos la representación de este Modelo de Optimización en los siguientes apartados:

- Consideraciones iniciales.
- Descripción de las Variables: Se definirán las variables a utilizar, así como su significado
- Descripción de las Restricciones: Por cada tipo de restricción, se mostrará su representación Matemática
- Descripción del Modelo completo: Se mostrará la representación del Modelo Matemático completo

3.2.1 Consideraciones iniciales.

Este modelo ataca al problema pensando en sus dos entidades básicas: agente y plan de trabajo, pero basándose en que un plan de trabajo no es más que un puesto(o una serie de puestos) y un grupo de descanso, aparecerán entonces restricciones que vienen expresadas en términos de éstas tres entidades : agente, trabajo y grupo de descanso.

Así pues, un trabajo no es más que el servicio realizado por un plan pero sin descansos, esto es, la colección de los puestos a realizar en un periodo determinado por el plan pero sin saber el grupo de descanso que toca.

La unión de un trabajo y un descanso constituye el plan a realizar.

3.2.2 Descripción de las Variables del Problema

Se tendrá, para cada Agente de Asignación a y Plan de Trabajo de Asignación p una variable x_{ap} . Esta variable cogerá el valor 1 si al Agente de Asignación a se le asigna el Plan de Trabajo de Asignación p , y 0 en otro caso. Así,

$$x_{ap} \in \{0, 1\}, \forall a \in A, \forall p \in P$$

Además, para cada Agente, crearemos una variable más $x'_a \in \{0, 1\}$ que valdrá 1 en el caso de que a no resulte asignado

3.2.3 Restricciones: Unicidad

Cada Agente de Asignación sólo podrá ser asignado a un Plan de Trabajo de Asignación

$$\forall a \in A, x'_a + \sum_{p \in P} x_{ap} = 1$$

3.2.4 Restricciones: Compatibilidades

Las restricciones de compatibilidad se representarán del siguiente modo.

$$x_{ap} = 0, \text{ si } \exists \text{ comp} \in C, a \in A, p \in P \text{ t.q. } \text{comp.Compatible}(a, p) = \text{NO}$$

3.2.5 Restricciones: Cardinalidades Puras

Las restricciones de Cardinalidad Puras se representarán del siguiente modo en función de si la cardinalidad se expresa sobre un conjunto de planes, de trabajos o de grupos de descanso:

$$\forall cp \in CP, \sum_{a \in A} \sum_{p \in cp.CONJ} x_{ap} \leq cp.MaxCardinalidad \text{ si } cp.CONJ \text{ es un conjunto de planes}$$

$$\forall cp \in CP, \sum_{a \in A} \sum_{t \in cp.CONJ} \sum_{p \in cp.CONJ(t)} x_{ap} \leq cp.MaxCardinalidad \text{ si } cp.CONJ \text{ es de trabajos}$$

$$\forall cp \in CP, \sum_{a \in A} \sum_{d \in cp.CONJ} \sum_{p \in cp.CONJ(d)} x_{ap} \leq cp.MaxCardinalidad \text{ si } cp.CONJ \text{ es de descansos}$$

3.2.6 Restricciones: Restricciones de Relación entre Planes de Trabajos

Las restricciones de Relación entre Planes de Trabajos se representarán del siguiente modo:

$$\forall r \in RPP \dots$$

Sea v_{ug} = Variable binaria que valdrá 1 si $\sum_{a \in AG} \sum_{pt \in rpp.P(u,g)} x_{ap} > 0$ (u, g) $\in RPP$ y 0 si ese sumando es cero, esto es la función característica asociada al número de asignaciones a los planes de la restricción.

Entonces la restricción lineal se expresa como..

$$\sum_{g \in G} v_{u'g} \leq (1 - v_{ug}) |G| \text{ donde } pt(u, g) \neq pt(u', g)$$

Esta linearización expresa claramente que si el lado izquierdo de la desigualdad es mayor que 0, entonces la variable que cuenta los planes asignados a la celda (u, g) tiene que ser 0 y al revés; si la celda (u, g) tiene planes asociados a su T-D y la celda (u', g) no tiene los mismos planes porque sus elementos T o D son diferentes entonces ninguno de esos planes debe ser asignado, esto es, $\sum_{g \in G} v_{u'g} = 0$.

3.2.7 Restricciones: Restricción de Relación entre Agentes

Esta restricción se representa como una Linearización, por lo que, para representarla, será necesario añadir las siguientes restricciones lineales:

$$\forall ra \in RA, \forall a_1, a_2 \in ra.A, a_1 \neq a_2, \forall cv \in CV_{a_1}^{ra} \sum_{p \in P(cv)} x_{a_1 p} \leq x'_{a_2} + \sum_{p \in P(rel(cv, a_1, a_2))} x_{a_2 p}$$

Notar que en cada una de estas restricciones estamos determinando que, si el lado izquierdo vale 1, entonces el lado derecho tiene que valer también 1. Si ahora examinamos el significado de cada uno de los dos lados, tendremos que:

- El lado izquierdo representa el número de veces que asignamos al Agente a_1 a un Plan de Trabajo p tal que $p \in P(cv)$
- El lado derecho representa, o bien la asignación del Agente a_2 al Plan de Trabajo NO_ASIGNADO (x'_{a_2}) o bien asignar al Agente a algún Plan de Trabajo de asignación perteneciente a los relacionados con p y a_1

Con esto representamos exactamente el significado de la restricción.

3.2.8 Descripción del Modelo de Optimización Completo

Así, el Modelo de Optimización completo quedaría de la siguiente forma:

$$\begin{aligned}
x'_a + \sum_{p \in P} x_{ap} &= 1 & \forall a \in A \\
x_{ap} &= 0 & \forall a \in A, p \in P, \exists c \in C \\
\sum_{a \in A} (\sum_{p \in cp} \text{CONJ} (x_{ap} \leq cp.MC)) & & \forall cp \in CP \\
\sum_{g \in G} v_{u'g} &\leq (1 - v_{ug}) |G| & \text{donde } pt(u, g) \neq pt(u', g) \\
\forall ra \in RA, \forall a_1, a_2 \in ra.A, a_1 \neq a_2, \forall cv \in CV_{a_1}^{ra} \sum_{p \in P(cv)} x_{a_1p} &\leq x'_{a_2} + \sum_{p \in P(\text{rel}(cv, a_1, a_2))} x_{a_2p}
\end{aligned}$$

3.3 Modelo Solver (siar::pt::ModeloConcertSolver)

El Modelo Solver está basado en la teoría de la Programación por Restricciones. En este sentido, cada una de las restricciones que aparecen en el Modelo de Asignación quedará representada como una restricción en el Modelo Solver.

Así, el procedimiento que vamos a seguir para describir este Modelo será el de:

1. Consideraciones iniciales.
2. Describir las variables existentes en el problema:
 - i. Descripción de las Variables fundamentales del problema (no de las auxiliares utilizadas en restricciones concretas)
 - ii. Descripción de las Restricciones utilizadas para mantener la coherencia entre estas variables
3. Descripción de la Modelización de cada una de las restricciones presentes en el Modelo Solver.

Se puede observar, pues, que los dos tipos de objeto que vamos a definir en este apartado son Variables y Restricciones. Para cada una de ellas, se informará de las siguientes características:

- **Variables**
 - *Tipo de Variable* (Entera, Conjunto de Enteros, etc.), junto con la clase Solver utilizada para su representación
 - *Significado de la Variable*: Una breve descripción de lo que significa la variable de cara al proceso de optimización
 - *Dominio de la Variable*: El conjunto de valores posibles que inicialmente contiene la variable
- **Restricciones**:
 - *Significado de la restricción*: Una breve descripción del sentido de la restricción con respecto al Modelo de Optimización
 - *Invariante*: Descripción de la propiedad que la restricción pretende mantener a lo largo de todo el árbol de búsqueda
 - *Métodos de Conservación del Invariante*:
 - Conservación Inicial: Descripción del proceso mediante el cual la restricción consigue que el invariante se respete al principio de la asignación
 - Conservación Dinámica: En función de lo que vaya ocurriendo a lo largo de la búsqueda de soluciones, descripción del proceso que permite a la restricción conservar la propiedad definida por el invariante
 - *Propiedades especiales de nuestra implementación*: Describiendo, en el caso de que la restricción haya sido implementada por nosotros, las propiedades especiales que conseguimos.

Con objeto de obtener *siempre* soluciones, vamos a crear un Plan de Trabajo Ficticio, llamado F , que representa la *no asignación* de un Agente.

En cualquier Modelo de Optimización escrito en ILOG Solver, no es tan importante la definición del Modelo como la calidad de las propagaciones contenidas en el mismo. Siendo esta calidad tan importante, dedicamos la siguiente sección a describir las técnicas de propagación utilizadas.

3.3.1 Técnicas de Propagación utilizadas

Conviene aquí introducir una serie de conceptos teóricos que serán útiles para definir la calidad de una restricción:

Definition 1. Dada una variable V , se dice que $D(V)$ es el conjunto de valores posibles que puede tomar dicha variable (es decir, su dominio). Cabe notar que este conjunto de valores puede variar dependiendo del momento de la búsqueda en que nos encontremos.

Definition 2. Una Restricción $R(V_1^R, \dots, V_n^R)$ sobre un conjunto de variables $\{V_1^R, \dots, V_n^R\}$ queda definida por un conjunto de t -uplas $T^R = \{T_1^R, \dots, T_m^R\}$ tales que $T_i^R = \{v_1^i, \dots, v_n^i\}$. Es decir, una Restricción sobre un conjunto de variables se define como un conjunto de t -uplas de valores para esas variables, y define que los valores que reciban esas variables en una solución deben corresponder a una de esas Tuplas

Definition 3. Se dice que una Restricción $R(V_1^R, \dots, V_n^R)$ es **Consistente** si existe al menos una t -upla $T \in T^R$ tal que $\forall i \in \{1, \dots, n\}, T_i \in D(V_i^R)$. Es decir, una Restricción es consistente si existe una solución que cumpla dicha restricción

Definition 4. Se dice que una Restricción $R(V_1^R, \dots, V_n^R)$ es **Arc-Consistent** si para todo $i \in \{1, \dots, n\}$, y para todo $v_i^j \in D(V_i^R)$ existe una tupla $T \in R$ tal que:

- $v_i^j = T_i$
- $T_k \in D(V_k^R)$, para todo $k \in \{1, \dots, n\}, k \neq i$

Es decir, que cada valor del dominio de cada variable de la restricción esté **justificado** por una t -upla.

Definition 5. Se dice que un Modelo de Optimización es **Arc-Consistent** si para toda variable V del Modelo, y para todo $v_i \in D(V)$, existe una t -upla T , tal que:

- $T \in R$, para todas las restricciones R del Modelo.
- $T_i = v_i$
- $T_k \in D(V_k)$, para todo $k \neq i$

Es decir, que para todo $v_i \in D(V)$, existe una solución factible del Modelo de Optimización en la que V tiene el valor v_i .

Se puede observar que si definimos un Modelo de Optimización que sea *Arc-Consistent*, entonces, independientemente del orden de instanciación de variables o valores, nunca tendríamos un *fallo* (fail) durante el proceso de búsqueda.

Aunque es en general **falso** que un Modelo de Optimización cuyas restricciones sean todas *Arc-Consistent* sea él mismo *Arc-Consistent*, es cierto que resulta muy interesante, con respecto a la velocidad de búsqueda, así como a la calidad de las soluciones encontradas, que todas ellas lo sean, o al menos se aproximen a serlo.

Así, y siempre que desde el punto de vista de la eficiencia resulta *rentable*, o bien usamos Restricciones existentes en ILOG Solver que mantengan estas propiedades, o bien construimos las nuestras propias de forma que, o bien se mantengan estas propiedades, o bien estemos próximos a mantenerlas. Para ello, utilizamos una técnica basada en *soportes*.

Definition 6. Se dice que un valor $v_i \in D(V)$ está **soportado** por la restricción $R(V_1^R, \dots, V_n^R)$ si:

- O bien R no afecta a la variable V .
- O bien R afecta a la variable V (digamos que es la variable con índice j) y existe una Tupla $T \in R$ tal que:
 - $T_j = v_i$, y
 - Para todo $k \in \{1, \dots, n\}, k \neq i, T_k \in D(V_k^R)$

Es decir, existe una solución factible con respecto a la restricción en la que la variable V toma el valor v_i .

Una restricción que esté implementada usando la técnica de soportes, tiene que garantizar que TODOS los valores existentes en los dominios de las variables a las que afectan están *soportados*, o lo que es lo mismo, es *arc-consistent*.

Para cada una de las restricciones que usen esta técnica, lo especificaremos y describiremos la forma en que esta propiedad queda mantenida.

3.3.2 Consideraciones iniciales.

Este modelo no ataca al problema pensando en sus dos entidades básicas: agentes y planes, sino que basándose en que un plan de trabajo no es más que un puesto(o una serie de puestos) y un grupo de descanso, la asignación se efectúa a través de tres entidades : agentes, trabajos y grupos de descansos.

Así pues, un trabajo no es más que el servicio realizado por un plan pero sin descansos, esto es, la colección de los puestos a realizar en un periodo determinado por el plan pero sin saber el grupo de descanso.

La unión de un trabajo y un descanso constituye el plan a realizar.

3.3.3 Variables del Problema

En el Modelo Solver, tendremos dos tipos fundamentales de variables:

- $x_a, \forall a \in A$: Representa el Trabajo asignado al Agente de Asignación a . Sus características son las siguientes:
 - *Tipo de Variable*: Es una variable de tipo Entera, representada con una instancia de la clase Solver **IloIntVar**.
 - *Significado de la Variable*: Esta variable representa el Plan de Trabajo de Asignación en bruto (sin descansos) que va a recibir el Agente a . Al ser de tipo Entera, en realidad contendrá el índice del Trabajo de Asignación asignado al Agente.
 - *Dominio de la Variable*: La variable contendrá inicialmente en su dominio los siguientes valores:
 - Todos los índices de los Trabajos de Asignación existentes en el Modelo de Asignación
 - Un índice especial, que representará el hecho de que el Agente no ha sido asignado (es decir, que ha sido asignado al Trabajo F Ficticio).
- $d_a, \forall a \in A$: Representa el Descanso asignado al Agente de Asignación a . Sus características son las siguientes:
 - *Tipo de Variable*: Es una variable de tipo Entera, representada con una instancia de la clase Solver **IloIntVar**.
 - *Significado de la Variable*: Esta variable representa el grupo de descanso que va a recibir el Agente a . Al ser de tipo Entera, en realidad contendrá el índice del Descanso asignado al Agente.

- *Dominio de la Variable:* La variable contendrá inicialmente en su dominio los siguientes valores:
 - Todos los índices de los Descansos existentes en el Modelo de Asignación
 - Un índice especial, que representará el hecho de que el Agente no ha sido asignado a ningún descanso(es decir, que ha sido asignado al *DF* Descanso Ficticio).
- $w_p, \forall p \in P \cup F$: Representa los Agentes de Asignación que han sido asignados a cada uno de los Trabajos, incluido *F*:
 - *Tipo de Variable:* Es una variable de tipo Conjunto de Enteros, representada con una instancia de la clase Solver **IloIntSetVar**.
 - *Significado de la Variable:* Representa los Agentes que están asignados al Trabajo *p*.
 - *Dominio de la Variable:* En un principio, la variable contendrá
 - En su Conjunto de valores Posibles, todos los Agentes de Asignación existentes
 - En su Conjunto de valores requeridos, el conjunto vacío.
- $v_d, \forall d \in D \cup DF$: Representa los Agentes de Asignación que han sido asignados a cada uno de los Descansos, incluido *DF*:
 - *Tipo de Variable:* Es una variable de tipo Conjunto de Enteros, representada con una instancia de la clase Solver **IloIntSetVar**.
 - *Significado de la Variable:* Representa los Agentes que están asignados al Descanso *d*.
 - *Dominio de la Variable:* En un principio, la variable contendrá
 - En su Conjunto de valores Posibles, todos los Agentes de Asignación existentes
 - En su Conjunto de valores requeridos, el conjunto vacío.
- $y_p, \forall p \in P \cup F$: Representa el número de Agentes de Asignación asignados a cada uno de los Trabajos, incluido *F*.
 - *Tipo de Variable:* Es una variable de tipo Entera, representada con una instancia de la clase Solver **IloIntVar**.
 - *Significado de la Variable:* Representa el número de Agentes que están asignados al Trabajo *p*. Equivale a la expresión $\text{IloCard}(w_p)$.
 - *Dominio de la Variable:* En un principio, la variable contendrá valores entre 0 y el número de Agentes de Asignación existentes.
- $z_d, \forall d \in D \cup DF$: Representa el número de Agentes de Asignación asignados a cada uno de los Descansos, incluido *DF*.
 - *Tipo de Variable:* Es una variable de tipo Entera, representada con una instancia de la clase Solver **IloIntVar**.
 - *Significado de la Variable:* Representa el número de Agentes que están asignados al Descanso *d*. Equivale a la expresión $\text{IloCard}(v_d)$.
 - *Dominio de la Variable:* En un principio, la variable contendrá valores entre 0 y el número de Agentes de Asignación existentes.

Para mantener la coherencia entre estos conjuntos de variables, se usa una restricción preexistente en ILOG Solver *IlcDistribute*, que garantiza, como mínimo, en todo momento, que:

- $\forall p \in P, y_{p.\min} = |\{a \in A: x_a = p\}|$
- $\forall p \in P, y_{p.\max} = |\{a \in A: p \in D(x_a)\}|$
- $\forall d \in D, z_{d.\min} = |\{a \in A: d_a = d\}|$
- $\forall d \in D, z_{d.\max} = |\{a \in A: d \in D(d_a)\}|$

La restricción *IlcDistribute*, en su modo de propagación *IlcExtended*, tiene una implementación *Arc-Consistent*.

3.3.4 Restricciones: Compatibilidades (siar::pt::CompatibilidadI)

Las compatibilidades, en Solver, se representan mediante la restricción de Compatibilidad. Esta restricción tiene las siguientes características:

- *Significado de la Restricción:* Esta restricción mantendrá la propiedad de que no existan en el dominio de ninguna variable x_a y d_a un Trabajo y Descanso que sea incompatible con el Agente de Asignación a y al revés, que el Agente de Asignación a no esté en el dominio de las variables w_p y v_d .
- *Invariante:* El invariante que se pretende mantener con esta restricción es el definido por las siguientes fórmulas:

$$\text{Si } \exists c \in C, a \in A, p \in T, t.q. c.\text{Compatible}(a, p) = \text{NO}, \text{ entonces } p \notin D(x_a)$$

$$\text{Si } \exists c \in C, a \in A, d \in D, t.q. c.\text{Compatible}(a, d) = \text{NO}, \text{ entonces } d \notin D(d_a)$$

Si $\exists c \in C, a \in A, \text{pt} \in \text{PT}, t.q. c.\text{Compatible}(a, \text{pt}) = \text{NO}$, entonces: $t \notin D(x_a)$ y $d \notin D(d_a)$ siendo $t = \text{trabajo de pt}$ y $d = \text{descanso de pt}$.

- *Métodos de Conservación del Invariante:* Este invariante sólo requiere de un proceso de conservación inicial, por lo que sólo se describirá este mismo.
 - Conservación Inicial: Para cada Agente de Asignación a se iterará sobre todos los valores contenidos en $D(x_a)$ y $D(d_a)$, y eliminará del mismo los valores de los Trabajos y Descansos tales que $\exists c \in C, a \in A, p \in T, t.q. c.\text{Compatible}(a, p) = \text{NO}$ o bien $\exists c \in C, a \in A, d \in D, t.q. c.\text{Compatible}(a, d) = \text{NO}$.
 - Conservación Dinámica: No procede.
- *Propiedades especiales de la restricción:* Se identifican dos propiedades interesantes en esta restricción:
 - Esta restricción se satisface durante la Conservación Inicial, por lo que es en sí *Arc-Consistent*, ya que cualquier valor de una variable que fuera susceptible de romper esta propiedad habría sido eliminado por este proceso
 - Si el Modelo de Optimización es en sí *Arc-Consistent*, añadir esta restricción no elimina esta propiedad del modelo, ya que la restricción en sí se limita a eliminar los valores imposibles desde el principio.

3.3.5 Restricciones: Cardinalidades Puras

Las Cardinalidades Puras se representan mediante una restricción simple sobre las cardinalidades de los Trabajos o Descansos. Así, si $cp \in CP$, se añadirá la siguiente restricción (en este caso lineal)

$$\begin{aligned} \sum_{p \in cp.CONJP} y_p &\leq cp.MaxCardinalidad \\ \sum_{d \in cp.CONJP} z_d &\leq cp.MaxCardinalidad \\ \sum_{pt \in cp.CONJP (p,d)=pt} y_p \cap z_d &\leq cp.MaxCardinalidad \end{aligned}$$

Siendo esta restricción Lineal, se añadirá a ILOG Solver, usando la propagación interna que éste tiene para este tipo de restricciones.

En cuanto a las *propiedades* de esta restricción, cabe definir dos casos:

- $|p.CONJP| = 1$: En este caso, se mantiene la propiedad de *Arc-Consistency*, ya que sencillamente estamos acotando el dominio de una variable (y_p), que a su vez está incluida en una restricción (*IlcDistribute*) que es *Arc-Consistent*. Añadiendo este tipo de restricciones no alteramos la propiedad de *Arc-Consistency*.
- $|p.CONJP| > 1$: En este caso, la restricción deja de ser *Arc-Consistent*, con lo que elimina esta propiedad del Modelo de Optimización, si este la tuviera.

3.3.6 Restricciones: Restricción de Relación entre Planes de Trabajos(siar::pt::CtRelacionPlanesDeTrabajoI)

La Restricción por Relación entre Planes de Trabajos se mantiene mediante una restricción *custom*, es decir, implementada por nosotros. Esta restricción tendrá las siguientes propiedades:

- *Significado de la Restricción*: Esta restricción (rpt) tendrá que satisfacer que dos cosas:
Si llamamos TD a la variable que informa de los descansos asociados a un trabajo y IPT a la variable unitaria que informa de si un plan de trabajo ha sido asignado o no, entonces..

Dado t Trabajo Si $\forall u, g$ (upla – columna de rpt) no existe un d : $(t, d) = (u, g)$ entonces : $d \not\subseteq D\{TD_t\}$ y $IPT[pt] = 0$ siendo $pt = (t, d)$.

$\sum_{pt \in rpt.PT(u,g)} IPT_{pt} \leq rpt.nbPT(u, g)$ (esto es, no se pueden asignar mas planes que el máximo que permita la celda de trabajos y descansos)

- *Invariante*: El invariante a conservar será el que expresa este último sumatorio, es decir, que no se asignen más planes de los que cualquier celda puede admitir.
- *Método de Conservación del Invariante*
 - Conservación Inicial: Si al comenzar la búsqueda no existe ningún Descanso $t.q d \in rpt.(u, g, t)$, eliminamos del dominio de la variable TD_t el valor d .
 - Conservación Dinámica: En cualquier momento en que tengamos que un agente a se asigna a un par de asignación $(t-d)$, actualizamos el valor $IPT_{pt} = 1$ siendo $pt=(t,d)$.
- *Propiedades especiales*: Esta restricción rompe la propiedad de *Arc-Consistency* del Modelo de Optimización.

3.3.7 Restricciones: Restricción de Relación entre Agentes (siar::pt::CtRestriccionRelacionAgentesPI)

La Restricción de Relación entre Agentes se representa en Solver mediante una restricción de tipo *custom*, es decir, una restricción implementada por nosotros. Así, procederemos a detallar las peculiaridades de la misma:

- *Significado de la Restricción*: Esta restricción tendrá que satisfacer que los Agentes de Asignación implicados en una Restricción de Relación entre Agentes sean asignados a Plan de Trabajos de Asignación que respeten dicha restricción, como queda explicado en la definición del Modelo de Asignación.
- *Invariante*: El invariante a conservar será el siguiente: Si existe un Agente $a \in \text{ra.AG}$, y existe $t \in \text{CV}_a^{\text{ra}}$ tal que para todo $p \in D(V_a)$, $p \in P(t)$, entonces, para cada $a' \in \text{ra.AG}$, $a' \neq a$, en $D(a')$ deben quedar sólo Plan de Trabajos pertenecientes a $\text{rel}(t, a, a')$.
- *Método de Conservación del Invariante*
 - *Conservación Inicial*: Si al comenzar a propagar existe un Agente $a \in \text{ra.AG}$, y existe $t \in \text{CV}_a^{\text{ra}}$ tal que para todo $p \in D(V_a)$, $p \in P(t)$, entonces, para cada $a' \in \text{ra.AG}$, $a' \neq a$, eliminamos de $D(a')$ todos los puestos que no pertenezcan a $\text{rel}(t, a, a')$.
 - *Conservación Dinámica*: En cualquier momento durante la búsqueda que se produzca que exista un Agente $a \in \text{ra.AG}$, y exista $t \in \text{CV}_a^{\text{ra}}$ tal que para todo $p \in D(V_a)$, $p \in P(t)$, entonces, para cada $a' \in \text{ra.AG}$, $a' \neq a$, eliminamos de $D(a')$ todos los puestos que no pertenezcan a $\text{rel}(t, a, a')$.
- *Propiedades especiales*: Esta restricción rompe la propiedad de *Arc-Consistency* del Modelo de Optimización, pero al ser una restricción de escasa aparición, y compleja de mejorar, nos limitamos a contar con la técnica de *Backtrack Inteligente*, contada con posterioridad en este documento, para solucionar los problemas que generara.

4 Algoritmos de Optimización

Los Algoritmos de Optimización son los encargados de resolver el Problema de Optimización producido por el Modelo de Asignación. Para ello, podrán hacer uso de Modelos de Optimización particulares para su técnica de resolución.

En particular, existen dos tipos de Algoritmos:

- *Algoritmos Genéricos*: Resuelven el Problema de Optimización en su totalidad, es decir, son capaces de tener en cuenta todos los Criterios de Evaluación existentes en el Modelo de Asignación
- *Algoritmos Específicos*: Resuelven un (o varios) tipos de Criterios de Evaluación, pero no todos ellos. Son normalmente usados por los Algoritmos Genéricos para resolver el Problema completo

4.1 Algoritmos Genéricos (siar::pt::AlgoritmoGenerico)

En la actualidad tenemos dos Algoritmos Genéricos:

- *Algoritmo Búsqueda Local* (siar::pt::AlgoritmoLSDescanso): Se usa cuando se desea implementar una búsqueda local que normalmente pretende mejorar una solución previamente obtenida.
- *Algoritmo Fixings* (siar::pt::AlgoritmoFixings): Se usa cuando existe algún Criterio de Evaluación por Criterio de Ordenación. Su peculiaridad es que, a la hora de comunicar información entre algoritmos de distinto tipo, utiliza los *Fixings*, que son restricciones sencillas que son asimilables por todos los algoritmos.

En el caso del algoritmo de Fixings, la técnica usada es corresponde a este proceso:

1. Seleccionar el siguiente Criterio de Evaluación ($ce := siguienteCriterio()$)
2. Seleccionar el Algoritmo Específico a utilizar ($alg := seleccionaAlgoritmoEspecifico(ce)$).
3. Con el algoritmo específico, resolver el Criterio de Evaluación seleccionado
4. Añadir restricciones al Modelo de Optimización que establezcan que el siguiente algoritmo *no puede empeorar la solución encontrada con respecto a éste Criterio de Evaluación* mediante el uso de *Fixings*.
5. Volver al paso 1, hasta que se acaben los Criterios de Evaluación.

4.1.1 Algoritmo Fixings (siar::pt::AlgoritmoFixings)

El Algoritmo Fixings se usa cuando tenemos de todos los tipos de Criterios de Evaluación. Por lo tanto, usará también varios tipos de Algoritmos Específicos. Esto crea un *problema de comunicación entre Algoritmos*, ya que éstos pueden ser de distinto tipo.

Para solucionar este problema, aparece el concepto de **Fixing**. Un Fixing es una restricción con las siguientes propiedades:

- *Es Simple*: Es decir, **todos** los Algoritmos Específicos son capaces de asimilarla sin bajar su rendimiento.
- *Determina que los siguientes algoritmos no empeoren la solución presente*, pero posiblemente sea aún más restrictivo. En este sentido, los Fixings añadidos podrán eliminar soluciones factibles, aunque, por contra, permitirán encontrar *buenas* soluciones de forma muy eficiente.

Existen en la actualidad distintos tipos de Fixing:

- FixingCubrirAgentePlan de Trabajos:
 - Contiene un Agente de Asignación y un conjunto de Planes de Trabajos de Asignación.
 - Determina que al Agente de Asignación sólo le podrá ser asignado uno de los Plan de Trabajos de Asignación contenidos en el conjunto.
- FixingCubrirAgenteDescansos:
 - Contiene un Agente de Asignación y un conjunto de Grupos de Descansos.
 - Determina que al Agente de Asignación sólo le podrá ser asignado uno de los Grupos de Descanso contenidos en el conjunto.
- FixingCubrirAgenteTrabajos:
 - Contiene un Agente de Asignación y un conjunto de Trabajos.
 - Determina que al Agente de Asignación sólo le podrá ser asignado uno de los Trabajos contenidos en el conjunto.
- FixingCubrirAgente:
 - Contiene un Agente de Asignación.
 - Determina que *este* Agente de Asignación será asignado
- FixingNoAsignarAgente:
 - Contiene un Conjunto de Agentes de Asignación.
 - Determina que *ningún* Agente de Asignación del conjunto será asignado

- **FixingCubrirPlan de Trabajo:**
 - Contiene un Conjunto de Plan de Trabajos de Asignación
 - Determina que *todos* los Plan de Trabajos de Asignación contenidos en el conjunto deben de ser cubiertos
- **FixingNoCubrirTrabajo**
 - Contiene un Conjunto de Trabajos.
 - Determina que *ninguno* de los Trabajos contenidos en el Conjunto debe de ser cubierto
- **FixingNoCubrirPlan de Trabajo**
 - Contiene un Conjunto de Plan de Trabajos de Asignación
 - Determina que *ninguno* de los Plan de Trabajos de Asignación contenidos en el Conjunto debe de ser cubierto
- **FixingCardinalidad:**
 - Contiene un Conjunto de objetos(Plan de Trabajos de Asignación-Trabajos o Descansos) y una Cardinalidad Máxima
 - Determina que el número máximo de veces que se deben cubrir los objetos contenidos en el conjunto es la Cardinalidad Máxima.
- **FixingCardinalidadPlan de TrabajosAgentes:**
 - Contiene un Conjunto de Agentes de Asignación y una Cardinalidad Máxima
 - Determina que el número máximo de veces que se deben asignar los Agentes de Asignación contenidos en el conjunto es la Cardinalidad Máxima.

Así, este algoritmo iterará sobre los Criterios de Evaluación existentes *por orden de prioridad* y:

1. En función del Criterio de Evaluación, seleccionará el Algoritmo a utilizar para su resolución
2. Comunicará al siguiente Algoritmo Específico, mediante un Fixing, la imposibilidad de empeorar el objetivo obtenido.

Para todo ello, va a hacer uso de las siguientes clases:

- *Algoritmo de Preprocesado de Compatibilidades*(`siar::pt::AlgoritmoPreproceso`): Es una clase encargada de instanciar de antemano un agente a aquél plan de trabajo para el que esta preasignado. Esto es, si resulta que el agente solamente es compatible con este plan, entonces se resuelve un algoritmo basado en Solver en donde se impondrá la restricción de que se instancie este agente a este plan.
Posteriormente y una vez que este algoritmo ha resuelto, si resulta satisfactoria la restricción se inpondrá un fixing de que el agente solo puede instanciarse a este plan y se propagará el fixing por todos los modelos de asignación para que cualquier otro algoritmo lo tenga en cuenta.
- *Selector Algoritmos* (`siar::pt::SeleccionadorAlgoritmo`): Es una clase que, en función de un Criterio de Evaluación, determina el Algoritmo que va a ser utilizado. En la actualidad el criterio para seleccionar el algoritmo será el siguiente:
 - Si el Criterio de Evaluación es por Coeficiente, se selecciona un Algoritmo Específico u otro en base a si se ha determinado que puede ser lineal o no. Ahora mismo, esta elección depende de ciertas condiciones y que han sido determinadas internamente.
*Ahora mismo se escoge un criterio de evaluación lineal si el número de restricciones de relación entre agentes y de planes de trabajo ambos es cero, además de si el tamaño del problema ($nbAgentes * nbPT * nbRestricciones$) es mayor de cero y menor de una constante: 5000000.*

- Si el Criterio de Evaluación es por Ordenación, se selecciona el Algoritmo Específico Solver por Ordenación.
- Si el Criterio de Evaluación es por Equilibrio, se selecciona un Algoritmo Específico de Equilibrio lineal o Algoritmo Específico de Equilibrio por Solver.
*Igual que antes, la elección de un algoritmo lineal o no, se basa en el cumplimiento del modelo de una serie de condiciones. Ahora mismo se escoge un criterio de evaluación lineal si el número de restricciones de relación entre agentes y de planes de trabajo ambos es cero, además de si el tamaño de las variables del problema ($nbAgentes * nbPT$) es mayor de cero y menor de una constante: .*
- *Generador Fixings (siar::pt::AlgoritmoFixingsBase):* Es la clase encargada de generar los fixings en función de la solución encontrada (además de otras funcionalidades) y del Criterio de Evaluación para el que se encontró dicha Solución. En la actualidad, se generan los siguientes Fixings:
 - *Si el Criterio de Evaluación es por Coeficiente sobre Agentes:* Si $a \in A$ es el Agente con menor coeficiente que ha sido asignado en la solución:
 - Se añadirá un Fixing de tipo FixingAsignarAgente para este agente.
 - *Si el Criterio de Evaluación es por Coeficiente sobre Planes de Trabajos:* Si $p \in P$ es el Plan de Trabajo con mayor coeficiente que ha sido asignado en la solución:
 - Se añadirá un Fixing de tipo FixingCardinalidad para todos los Planes con coeficiente *igual* o menor al de p , y Cardinalidad Máxima igual al número de veces que estos planes han sido asignados en la solución.
 - *Si el Criterio de Evaluación es por Coeficiente sobre Agentes - Plan de Trabajos:* Para cada $a \in A$ y $p \in P$ asignado en la solución:
 - Se añadirá un Fixing de tipo FixingCubrirAgentePlanesDeTrabajo para el Agente a y todos los Planes con coeficiente *igual* o menor al de p en la asignación.
 - *Si el Criterio de Evaluación es por Ordenación:* Para cada $a \in A$, si $p_a \in P$ es el Plan de Trabajo de Asignación asignado a a , entonces se creará un fixing de tipo FixingCubrirAgentePlandeTrabajo que contenga todos los Planes de Trabajos de Asignación que a prefiere *igual* que p_a , es decir, todos los planes que sean *iguales* a p_a con respecto a las preferencias de a .
 Además, y dado que un plan no es más que un trabajo y un descanso, entonces de forma paralela se añaden fixings de tipo FixingCubrirAgenteTrabajo y FixingCubrirAgenteDescanso.
 - Si el Criterio de Evaluación es por Equilibrio: Para cada $a \in A$ y $d \in D$ (descanso) asignado en la solución, como actualmente este criterio se usa para equilibrar posteriormente los grupos de descanso:
 - Se añadirá un Fixing de tipo FixingCubrirAgente para el Agente a asignado en la solución.
 - Se añadirá un Fixing de tipo FixingCubrirAgenteDescanso para el Agente a y todos los Planes con descanso *igual* al d en la asignación.

4.2 Algoritmos Específicos

Los algoritmos específicos son Algoritmos especializados en la resolución del Modelo de Asignación para uno o varios Criterios de Evaluación, pero no para todos ellos. Esto nos permite resolver dichos criterios de forma muy eficiente, pero nos hace perder visión sobre el Modelo de Asignación visto de forma Global, por lo que estos algoritmos siempre son utilizados desde otros Algoritmos Genéricos que sí tienen esta visión.

Cada Algoritmo específico utiliza un Modelo de Optimización, pudiendo ser dicho Modelo distinto para cada algoritmo. En la descripción de cada uno de los Algoritmos Específicos existentes se determinará el Modelo de Optimización que utilizan.

En la actualidad tenemos varios Algoritmos Específicos:

- Algoritmo CPLEX por Coeficiente (siar::pt::lineal::AlgoritmoCEACPlanDeTrabajoLineal): Este Algoritmo está basado en la tecnología ILOG CPLEX, y resuelve los Criterios de Evaluación por Coeficiente en los que se usa CPLEX.
- Algoritmo CPLEX por Equilibrio (siar::pt::lineal::AlgoritmoEquilibrioLineal): Este Algoritmo está basado en la tecnología ILOG CPLEX, y resuelve los Criterios de Evaluación por Equilibrio lineales.
- Algoritmo Solver por Equilibrio (siar::pt::descanso::AlgoritmoEquilibrioSolver): Este Algoritmo está basado en la tecnología ILOG Solver, y resuelve los Criterios de Evaluación por Equilibrio no lineales.
- Algoritmo Solver por Coeficiente: Este Algoritmo está basado en la tecnología ILOG Solver, y resuelve los Criterios de Evaluación por Coeficiente en los que no se usa CPLEX. En función del tipo de atributo manejado por el criterio de evaluación (agente, plan o agente-plan) se utiliza AlgoritmoCEACAgente, AlgoritmoCEACAgentePlanDeTrabajo o AlgoritmoCEACPlanDeTrabajo.
- Algoritmo Solver por Ordenación (siar::pt::descanso::AlgoritmoCriterioOrdenacion): Este Algoritmo está basado en la tecnología ILOG Solver, y resuelve los Criterios de Evaluación por Ordenación.

A continuación se describen en más detalle ambos algoritmos.

4.2.1 Algoritmo CPLEX por Coeficiente (siar::pt::AlgoritmoCEACPlanDeTrabajoLineal)

Este Algoritmo, mediante el uso de un Modelo de Optimización basado en ILOG CPLEX (ModeloConcertCplex), establecerá una función objetivo Lineal representando los costes de realizar cada una de las posibles asignaciones entre Agentes y Planes de Trabajos.

Existen dos casos de función objetivo, en función de si el Criterio de Evaluación tiene en cuenta la Reserva Enumerada o no la tiene en cuenta:

- En caso de que no la tenga en cuenta, la Función Objetivo quedará como sigue:

$$\sum_{a \in A} \sum_{p \in P} (x_{ap} * C^c.coef(a, p))$$

- En caso de que sí se tenga en cuenta, la Función Objetivo quedará como sigue

$$\sum_{a \in A} \sum_{p \in P} (x_{ap} * C^c.coef(a, p)) + \sum_{p \in RE} (C^c.offset * \sum_{a \in A} (x_{ap}))$$

4.2.2 Algoritmo CPLEX por Equilibrio (siar::pt::lineal::AlgoritmoEquilibrioLineal)

Este Algoritmo, mediante el uso de un Modelo de Optimización basado en ILOG CPLEX (ModeloConcertCplex), establecerá una función objetivo Lineal cuadrática representando los costes de realizar cada una de las posibles asignaciones entre Agentes y Planes de Trabajos.

Existen dos casos de función objetivo, en función de si el Criterio de Evaluación tiene en cuenta la Reserva Enumerada o no la tiene en cuenta:

- En caso de que no la tenga en cuenta, la Función Objetivo quedará como sigue:

$$\{\min\} \sum_{v \in V} \left(\sum_{AC.(a,p)=v} x_{ap} \right)^2$$

siendo $V = \{v \in C^c.coef(a, p): a \in A, p \in PT\}$

- JMP: No se tiene en cuenta. En caso de que sí se tenga en cuenta, la Función Objetivo quedará como sigue

4.2.3 Algoritmo Solver por Equilibrio (siar::pt::descanso::AlgoritmoEquilibrioSolver)

Este Algoritmo, mediante el uso de un Modelo de Optimización basado en ILOG Solver (ModeloConcertSolver), establecerán una heurística no lineal para realizar cada una de las posibles asignaciones entre Agentes y Planes de Trabajos.

Notar que dicho algoritmo se basa en un criterio de evaluación de equilibrio por coeficiente basado en un atributo calculado.

Su heurística es la siguiente:

1. Escoge a aquél agente de asignación a con menor número de planes posibles a asignar, esto es, a aquél cuya suma de dominios de las variables de trabajo y descanso sea menor. Habiendo varios se escogerá por el orden de instanciación.
2. Para este agente a se le asigna el plan de trabajo p:
 - cuyo valor a través del atributo calculado AC.coef(a,p) haya aparecido un menor número de veces en este momento. Para ello se emplea un array de enteros reversibles que informan del número de veces que ha aparecido cada valor posible del atributo calculado.
 - de forma que la asignación sea factible (en el sentido de que se cumplan todas las restricciones)
3. Se vuelve al paso 1.

4.2.4 Algoritmo Solver por Coeficiente (siar::pt::descanso::AlgoritmoCEACAgenteDescansos, siar::pt::descanso::AlgoritmoCEACAgentePlanDeTrabajoDescansos, siar::pt::descanso::AlgoritmoCEACPlanDeTrabajoDescansos)

Estos Algoritmos, mediante el uso de un Modelo de Optimización basado en ILOG Solver (ModeloConcertSolver), establecerán una heurística no lineal para realizar cada una de las posibles asignaciones entre Agentes y Planes de Trabajos.

Veamos en primer lugar, como funciona el AlgoritmoCEACAgenteDescansos que se basa en un criterio de evaluación por coeficiente basado en un atributo calculado de Agentes:

1. Ordena a los agentes en base a su mejor coste para todos los planes de trabajo, esto es, si tenemos dos agentes a y a'...t.q
 $C^c.coef(a) < C^c.coef(a')$ entonces a es mejor que a' y se asignaría en primer lugar.
2. Una vez ordenados los agentes se procede a su instanciación, en base a este orden y teniendo en cuenta si no ha sido ya asignado.
3. Para un determinado agente se le asigna el primer plan de trabajo que hace que el número de agentes que se van a quedar sin asignar por su instanciación no aumente; esto se hace imponiendo la condición de que la variable asociada al plan dummy que registra la cardinalidad de agentes no asignados sea menor o igual que su valor mínimo actual.

4. Se volvería al paso 2.

De forma muy parecida funciona el AlgoritmoCEACAgentePlanDeTrabajoDescansos que se basa en un criterio de evaluación por coeficiente basado en un atributo calculado de Agente-Plan:

1. En este algoritmo se ordena a los planes en base a su mejor coste para todos los agentes, esto es, si tenemos dos planes de trabajo p y $p' \dots t.q$

Para todo Agente a , si $C^c.coef(a,p) < C^c.coef(a,p')$ entonces p es mejor que p' y se asignaría en primer lugar.

Reseñar que en dicho coste se ha tenido en cuenta la posibilidad de que el plan fuera reusable y por consiguiente su coste varía.

2. Una vez ordenados los planes se procede a su instanciación, en base a este orden y teniendo en cuenta tres cosas:
 - si no ha sido ya asignado.
 - si el tamaño de los agentes posibles a los que puede asignarse el plan es menor.
 - si existe al menos una pareja (a,p) que es válida (esto es, que verifica todas las restricciones).
3. Para ese plan p escogido en la etapa anterior se le asigna finalmente el primer agente que hace que la pareja (a,p) sea válida (en el sentido explicado anteriormente) y cuyo coste sea el menor.
4. Se volvería al paso 2.

Finalmente el AlgoritmoCEACPlanDeTrabajoDescansos que se basa en un criterio de evaluación por coeficiente basado en un atributo calculado de Plan:

1. Ordena a los planes en base a su mejor coste para todos los agentes, esto es, si tenemos dos planes de trabajo p y $p' \dots t.q$
 $C^c.coef(p) < C^c.coef(p')$ entonces p es mejor que p' y se asignaría en primer lugar.
2. Una vez ordenados los planes se procede a su instanciación, en base a este orden y teniendo en cuenta si no ha sido ya asignado y si el dominio de agentes a los que pueda asignarse es el menor.
3. Para un determinado plan se le asigna el primer agente que verifique:
 - que no ha sido ya asignado.
 - que el tamaño de los posibles planes a los que puede asignarse el agente sea el más pequeño.
4. Se volvería al paso 2.

4.2.5 Algoritmo Solver por Ordenación (siar::pt::descanso::AlgoritmoCriterioOrdenacion)

Este Algoritmo plantea una heurística con objeto de realizar una asignación óptima en cuanto al Criterio de Evaluación por Ordenación se refiere. Se puede observar que, dado el caso en que **todos** los Plan de Trabajos de Asignación tuviesen exactamente la misma lista de Agentes de Asignación ordenados con respecto a las prioridades, se podría utilizar la siguiente heurística:

1. Ordenamos los Agentes de Asignación en función de su orden en la lista única existente

2. Vamos asignando un Plan de Trabajo de Asignación a cada Agente por el orden inferido. Cada vez que se asigna uno de estos Plan de Trabajos de Asignación, se *propagan* las restricciones de modo que se eliminen los Plan de Trabajos de Asignación que ya no son asignables.

De esta forma, garantizaríamos que, siempre que se produjera un conflicto, este sería *justificable*, ya que si el Agente que entra en conflicto no puede obtener un Plan de Trabajo de Asignación es por una de estas dos razones:

- O bien un Agente de Asignación más prioritario se lo ha llevado antes
- O bien el hecho de que un Agente más prioritario se llevara otro Plan de Trabajo de Asignación produjo que el Plan de Trabajo de Asignación preferido no se haya podido asignar.

En cualquiera de estos dos casos, se consideraría que el conflicto es *justificable*, por lo que no sería un problema a la hora de determinar la *bondad* de la solución.

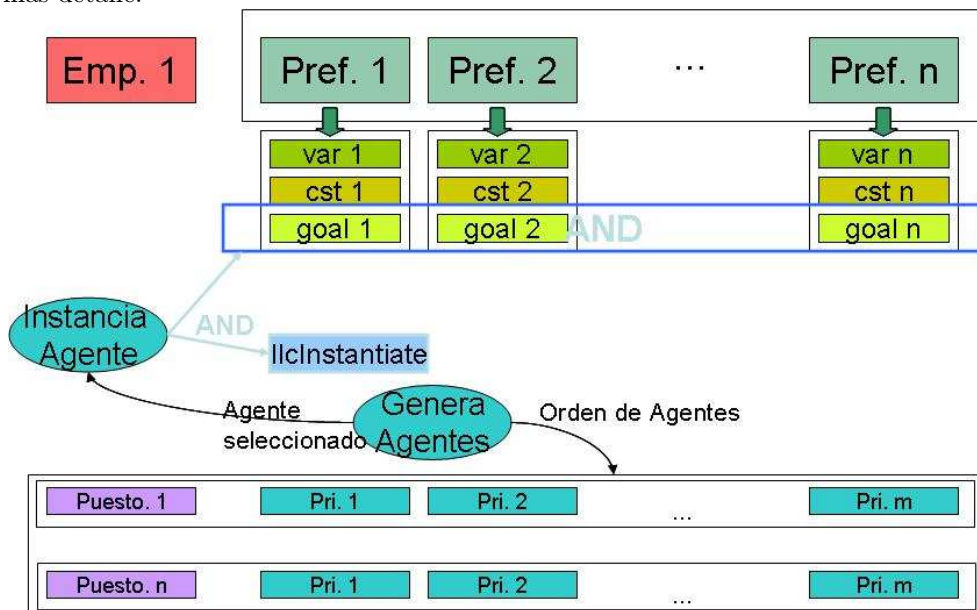
La realidad es, en general, otra, que hace que esta técnica no funcione directamente. Ocurren dos cosas que lo impiden:

- No siempre las Listas de Agentes de Asignación para los Plan de Trabajos de Asignación son iguales. Es decir, dependiendo de la naturaleza del Plan de Trabajo de Asignación, pueden aparecer Listas de Agentes distintas.
- Aunque es importante no generar conflictos en la solución, es más importante el no dejar Agentes de Asignación sin asignar, por lo que, en cualquier caso, la solución ofrecida por este algoritmo no será óptima si deja Agentes de Asignación sin Asignar.

Por ello, es necesario tomar medidas que gestionen estos dos problemas, usando de todas formas un algoritmo heurístico. Estas medidas son:

- Saltos atrás *inteligentes* a lo largo de la búsqueda, para solucionar el problema de las listas de Agentes distintas (y eventualmente para solucionar el problema de los Agentes no asignados)
- Propagación más potente, para solucionar el problema de los Agentes no Asignados.

Todas estas técnicas serán expuestas posteriormente. Por ahora, se introduce un gráfico que muestra la técnica seguida para solucionar el Modelo de Optimización y, a continuación, se entrará en más detalle.



4.2.2.1 Algoritmo de Búsqueda (Genérico)

En general, se puede dividir el Algoritmo de Búsqueda en dos fases principales:

- **Ordenación de los Agentes** de Asignación con respecto a las Prioridades de los Plan de Trabajos
- **Instanciación de cada Agente** de Asignación con respecto a sus Preferencias

4.2.2.1.1 Ordenación de los Agentes

El objetivo de esta fase consiste en conseguir una ordenación única de Agentes de Asignación de modo que la Asignación que conseguiríamos con el método explicado al principio de esta sección genere el menor número de conflictos. De este modo, las técnicas de reparación tendrán un impacto mínimo sobre el rendimiento del Algoritmo.

Así, el proceso se divide en dos pasos:

- Conseguir una colección de conjuntos de Planes de Trabajos de Asignación con ordenaciones similares.
- Mezclar las ordenaciones producidas por dichos conjuntos de forma que sea suficientemente buena en el sentido expresado con anterioridad.

Para conseguir una colección de conjuntos de Planes de Trabajos de Asignación se sigue el siguiente algoritmo:

1. Sea pri la primera prioridad del conjunto de Prioridades contenido en el Criterio de Evaluación
 - i. Hacemos $pri_actual := pri$
 - ii. Hacemos $conj_inicio := P$, siendo P el conjunto de todos los Plan de Trabajos de Asignación
 - iii. Hacemos $C := \{conj_inicio\}$
2. Mientras $pri_actual \neq nil$
 - i. Para cada conjunto $conj$ perteneciente a C
 - a) Si pri_actual induce distintas ordenaciones $(O_1, ..., O_n)$ sobre los Plan de Trabajos de $conj$, entonces creamos tantos conjuntos de Plan de Trabajos $(C_1, ..., C_n)$ como ordenaciones, siendo C_i el conjunto de Plan de Trabajos de Asignación pertenecientes a $conj$ tales que pri_actual induce la ordenación O_i sobre el Plan de Trabajo.
 - ii. Hacemos $C := \{C_1, ..., C_n\}$
 - iii. Hacemos $pri_actual := pri_actual.siguiente$.
3. Devolvemos la colección C resultante.

Sabemos ahora que esta colección de Conjuntos de Plan de Trabajos de Asignación en realidad nos está representando una colección de Ordenaciones *distintas* de Agentes de Asignación con respecto a Plan de Trabajos de Asignación para conjuntos de Plan de Trabajos de Asignación. Ahora tenemos que inferir una Ordenación de Agentes a partir de estas Ordenaciones inferidas. Para ello, disponemos en la actualidad de dos técnicas:

- O bien escogemos directamente la Ordenación cuyo Conjunto de Planes de Trabajos de Asignación C_i tenga una Cardinalidad mayor, de modo que estamos escogiendo la ordenación *más representativa*.
- O bien damos un valor a cada Agente a igual a $\sum_{i \in \{0, ..., n\}} posicion(a, O_i)$, y ordenamos los Agentes por este valor, de modo que estamos escogiendo la ordenación *más equitativa*.

Una vez conseguida esta lista ordenada de Agentes, procedemos a la instanciación de cada uno de ellos en función de sus preferencias, en el orden marcado por la Lista. De esto se encarga el Goal *GenerarAgentesI*, cuya estructura es de este estilo:

```
IlcGoal GenerarAgentesI::execute() {
    Agente *agente = NULL;

    // Seleccionar siguiente Agente sin instanciar, siguiendo el orden
    // inferido y guardarlo en 'agente'

    return IlcAnd(InstanciarAgente(getSolver(), _algoritmo, agente),
                  this);
}
```

La aparición de las prioridades implicadas (ordenaciones de agentes con respecto a planes que dependen de otros planes. Véase por ejemplo la prioridad de agentes por baremo de descanso) hace necesario modificar ligeramente esta heurística en la elección del conjunto de planes (2.i) C_i asociada a la ordenación O_i .

Para ello se sabe que el conjunto de planes da la misma ordenación de agentes teniendo en cuenta el tipo de ordenación (inferido creciente, enumerado..), pero en la práctica el orden real puede ser diferente, pues dicho orden puede depender de una instanciación de algún plan anterior y eso no se contemplaba hasta ahora.

Lo que se hace entonces es que en la etapa 2.i, una vez obtenidos los grupos C_i se eligen subgrupos aleatorios de planes C_{ij} contenidos en C_i y se calculan las ordenaciones de los agentes con respecto a estos subgrupos de planes O_{ij} . Aquella ordenación O_{ik} más representativa será la tenida en cuenta para este grupo C_i y el tamaño del mismo será $|C_{ik}|$ necesario para efectuar la mezcla de la segunda etapa que elige la mejor ordenación entre todos los conjuntos de planes.

4.2.2.1.2 Instanciación de cada Agente

Instanciar un Agente consiste en asignarle un Plan de Trabajo de Asignación, o bien decidir que se queda como NO_ASSIGNADO. Esta asignación se realiza de forma indirecta, a través de las preferencias. Así, se van seleccionando las Preferencias del Agente, y, por cada una de ellas, dependiendo de su tipo, se crea una *Variable Restringida*, una *Restricción* y un *Goal*. A continuación se describe cada uno de estos objetos:

- La *Variable Restringida* que se crea depende del modo en que se expresa la preferencia del Agente:
 - Si la preferencia es *Inferida*, se crea una *Variable Restringida* cuyo dominio es el resultado de llamar al método *getValoresPosibles* de la preferencia más el valor de la preferencia dummy.
 - Si la preferencia es *Enumerada*, se crea una *Variable Restringida* cuyo dominio son todos los valores de la preferencia del Agente más el valor de la preferencia dummy.
 - Si la preferencia es *Enumerada por Conjuntos*, se crea una *Variable Restringida* cuyo dominio es la unión de los Conjuntos que expresan la preferencia del Agente más el valor de la preferencia dummy.
- La *Restricción* que se crea sencillamente mantiene la relación entre los valores de la *Variable Restringida* y los posibles Planes de Trabajos de Asignación (Trabajo y Descanso) para asignar al Agente. Esta restricción queda definida por:
 - *Invariante*:
 - Si no existe ningún trabajo en el dominio del Agente tal que el valor de la Preferencia con respecto a ese trabajo sea v , entonces v no puede estar en el dominio de la *Variable Restringida* representando a la preferencia

- Si no existe ningún descanso en el dominio del Agente tal que el valor de la Preferencia con respecto a ese descanso sea v , entonces v no puede estar en el dominio de la *Variable Restringida* representando a la preferencia
- Si no existe el valor v en el dominio de la *Variable Restringida* que representa la preferencia del Agente, entonces no puede existir ningún Trabajo-Descanso en el dominio del Agente cuyo valor con respecto a la preferencia sea v .
- *Métodos para mantener el invariante:*
 - Mantenimiento Inicial (post): Inicialmente, se revisan los valores de los dominios tanto de la preferencia como del Agente, y se eliminan los que son inconsistentes
 - Mantenimiento Dinámico (propagate): El Mantenimiento dinámico se hace mediante un sistema de *soportes*. Este sistema consiste en mantener una justificación (soporte) para cada uno de los valores presentes en el dominio de la preferencia. Una *justificación* para un valor v del Dominio de la preferencia es un Plan de Trabajo p (trabajo-descanso) en el dominio del Agente cuyo valor con respecto a la preferencia sea v . Para conseguir esto, se realizan dos acciones:
 1. Cuando se elimina un Trabajo p del dominio de Trabajos del Agente, se chequea si este Trabajo era soporte de algún valor v de la preferencia.:
 - i. Si es así, se busca otro Trabajo p' en el dominio de Trabajos del Agente cuyo valor con respecto a la preferencia sea v :
 - a) Si existe este Trabajo p' , se pone como soporte de v , y se continúa
 - b) Si no existe este Trabajo, se elimina v del dominio de la preferencia y se continúa
 - ii. Si no, no se hace nada
 2. Cuando se elimina un valor v del dominio de la preferencia, se eliminan todos los Trabajos de Asignación del dominio del Agente tales que su valor con respecto a la preferencia sea v .
De igual manera se sigue con los descansos.
- El *Goal* que se crea pretende instanciar la *Variable Restringida* asociada a la preferencia, en el orden de valores preferido por el Agente. Así, dependiendo de la forma de expresar la preferencia, tenemos tres Goals:
 - *InstanciaVariableInferido*: Este Goal se encarga de instanciar la Variable Restringida asociada a la preferencia de por un orden inferido

```

IlcGoal InstanciaVariableInferidoI::execute() {
    IlcInt valPref = IloIntMax;

    if ( !_varPref.isBound() ) {
        // Seleccionamos, el valor más pequeño o más grande posible de
        // la Variable Restringida representando a la preferencia, en
        // en función de si la preferencia del Agente es por
        // ORDEN_CRECIENTE o ORDEN DECRECIENTE, respectivamente

        return IlcOr ( _varPref == valPref,
                       IlcAnd(_varPref != valPref, this) );
    }
}

```

- *InstanciaVariableEnumerado*: Este Goal se encarga de instanciar la Variable Restringida asociada a la preferencia a partir de una enumeración de valores

```
IlcGoal InstanciaVariableEnumeradoI::execute() {
    IlcInt valPref = IloIntMax;

    if ( !_varPref.isBound() ) {
        // Seleccionamos el primer valor de la preferencia del
        // Agente que esté en el dominio de la variable _varPref

        return IlcOr ( _varPref == valPref,
                       IlcAnd(_varPref != valPref, this) );
    }
}
```

- *InstanciaVariableEnumeradoConjuntos*: Este Goal se encarga de instanciar la Variable Restringida asociada a la preferencia a partir de una enumeración de Conjuntos de valores.

```
IlcGoal InstanciaVariableInferidoEnumeradoConjuntosI::execute() {
    IlcIntArray conjPref;

    if ( !_varPref.isBound() ) {
        // Seleccionamos el primer conjunto de la lista de conjuntos
        // Agente que esté en el dominio de la variable _varPref

        return IlcOr ( IlcMember(_varPref, conjPref),
                       IlcAnd(IlcNotMember(_varPref, conjPref),
                              this) );
    }
}
```

En última instancia, por si después de evaluar todas las preferencias quedara más de un Plan de Trabajo compatible con todas las elecciones realizadas, se ejecuta un *IlcInstantiate* sobre la variable de los Planes de Trabajos (Trabajos y Descansos) del Agente.

Estos Goals, restricciones y variables representan la heurística básica que se define al principio de esta sección. A este respecto hemos identificados dos problemas fundamentales:

1. Existencia de ordenaciones distintas de Agentes de Asignación en función de características del Plan de Trabajo de Asignación concreto (perfil FSC, Escalafón condicionado, etc.,): Para este caso, utilizamos técnicas de *Backtrack Inteligente*, que se describirán posteriormente.
2. Dominancia del Criterio de no dejar Agentes sin asignación. Utilizamos dos técnicas distintas para resolver este problema:
 - i. Técnicas de propagación más completas
 - ii. *Backtrack Inteligente*

En las siguientes dos secciones se describirán las técnicas utilizadas. Cabe decir, en cualquier caso, que estas dos técnicas son en cierto modo equivalentes. Es decir, en el momento en que se realiza un *Backtrack Inteligente* es porque la propagación anterior no ha sido capaz de detectar que este *Backtrack* iba a hacerse, y una propagación más completa lo hubiese evitado.

4.2.2.2 Técnica de *Backtrack Inteligente*

Durante el proceso de asignación, en cualquiera de los Goals que se ejecutan, podemos encontrarnos con una situación *incorrecta*. En la actualidad nos encontramos con la siguiente situación posible:

- Existe un Plan de Trabajo que el Agente A_1 , que se está instanciando actualmente, prefiere a cualquiera de los que se le pueden asignar en este momento, y este Plan de Trabajo ha sido asignado a otro Agente A_2 que fue instanciado con anterioridad en este proceso de Asignación. Resulta, además, que A_1 es prioritario a A_2 para ese Plan de Trabajo.

En este caso, intentaríamos deshacer la decisión que llevó a esta incorrección y continuar la búsqueda.

Notar que aunque estemos hablando de plan de trabajo la técnica es extrapolable a sus variables más primarias: trabajo y descanso.

4.2.2.2.1 Técnica de Backtrack Inteligente: Prioridades

A la hora de introducir esta técnica es conveniente introducir serie de objetos que van a entrar en juego:

- Llamaremos A al Agente que estamos asignando en la actualidad
- Llamaremos P_1, \dots, P_m a las preferencias tenidas en cuenta en el Criterio de Evaluación
- A_1, \dots, A_n serán los Agentes, ordenados por orden de instanciación, que han sido asignados *antes* que A en este proceso de asignación
- Llamaremos $L(A_i, P_j)$ al Label asignado al Punto de Decisión (IlcOr) que se ejecutó al instanciar la preferencia P_j para el Agente A_i .
- Llamaremos $L(A_i)$ al Label asignado al Punto de Decisión (IlcOr) que se ejecutó al llamar al *IlcInstantiate* sobre el Agente A_i .

Dicho esto, en *todos* los Goals de instanciación de preferencias se incluirá un chequeo previo como el que sigue, suponiendo que estamos instanciando la preferencia P_i :

1. Hacer $A_S := \text{nil}$ (A_S es el Agente Seleccionado para hacer el Backtrack)
2. Hacer $V_S^P :=$ el valor de la preferencia P_i que el Agente A prefiere entre todos los que se le pueden asignar en este momento.
3. Para cada Agente $A' \in \{A_n, \dots, A_1\}$ (notar el orden inverso):
 - i. Obtenemos el Plan de Trabajo P' que le fue asignado al Agente A' .
 - ii. Si:
 - a) El Plan de Trabajo P' es compatible en todos los sentidos al Agente A . En este sentido debe de ser compatible con respecto a todas las restricciones definidas en el Modelo.
 - b) El Plan de Trabajo P' tiene los mismos valores con respecto a las preferencias P_1, \dots, P_{i-1} que los seleccionados por el Agente A hasta la preferencia P_i (que es la que estamos instanciando)
 - c) El valor del Plan de Trabajo P' con respecto a la preferencia P_i es preferido a V_S^P por el Agente A .
 - d) A es más prioritario que A' para el Plan de Trabajo P' .

Entonces hacemos:

- a) $A_S := A'$
 - b) $V_S^P := P_i.\text{valor}(P')$
- iii. Si no, continuamos al Agente anterior en la lista.
4. Si $A_S \neq \text{nil}$, entonces hacemos un Backtrack al Label $L(A_S, P_i)$.

En resumen, lo que hacemos con esta técnica de backtrak es, cada vez que decidimos una preferencia para un Agente, ver si existiese otro Agente que hubiera sido asignado antes que este y al que le hubiéramos asignado un Plan de Trabajo que el nuestro prefiriera a cualquiera de los que le podemos dar ahora, y además fuera más prioritario para el mismo.

De entre todos los Agentes que cumplen esta propiedad, elegiremos el que tenga el Plan de Trabajo que más prefiera, y de entre ellos, el que haya sido asignado el último.

De esta manera, evitamos los conflictos que pudieran haber sido generados por una ordenación incorrecta de los Agentes.

4.2.2.2.2 Técnicas de Propagación utilizadas

Con respecto a la parte de Búsqueda, y con objeto de propagar mejor, hacemos dos cosas:

- Implementamos las restricciones adicionales usando técnicas de propagación eficientes, como es el caso de la técnica de *soportes* utilizada en la restricción que mantiene la correspondencia entre los valores de las preferencias y los de los Plan de Trabajos asignados a los Agentes (descrita con anterioridad)
- Usamos técnicas de *mirar hacia adelante*.

Las técnicas de mirar hacia adelante consisten básicamente en, a la hora de decidir el valor que se asigna a una variable (ya sea esta la correspondiente a una Preferencia o a un Plan de Trabajo), se prueban varias posibilidades, examinando sus efectos y decidiendo a posteriori.

En nuestro caso, utilizamos estas técnicas con objeto de ver si la asignación que produzco provoca que se deje sin asignar a un Agente.

4.2.2.2.4 Conclusiones

Con esto, los Goals que creamos para instanciar las preferencias quedarían como sigue:

- *InstanciaVariableInferido*

```
IlcGoal InstanciaVariableInferidoI::execute() {
    IlcInt valPref = IloIntMax;

    // Chequeamos si podemos hacer un backtrack inteligente, ya sea
    // por una preferencia o por un NO_ASIGNADO. En caso de poder
    // hacerlo, se salta desde aquí, así que la ejecución de este
    // Goal se interrumpe.

    if ( !_varPref.isBound() ) {
        // Para cada valor de '_varPref', ordenado por preferencia del
        // Agente, chequeamos si produce que otro Agente se quede sin
        // asignar:
        // - Si la respuesta es NO, seleccionamos este valor
        // - En otro caso, continuamos
        // Si después de intentarlo con todos los valores, todos ellos
        // dejan algún Agente sin asignar, elegimos el más preferido.

        return IlcOr ( _varPref == valPref,
                       IlcAnd(_varPref != valPref, this), label );
    }
}
```

- *InstanciaVariableEnumerado*

```

IlcGoal InstanciaVariableEnumeradoI::execute() {
    IlcInt valPref = IloIntMax;

    // Chequeamos si podemos hacer un backtrack inteligente, ya sea
    // por una preferencia o por un NO_ASIGNADO. En caso de poder
    // hacerlo, se salta desde aquí, así que la ejecución de este
    // Goal se interrumpe.

    if ( !_varPref.isBound() ) {
        // Para cada valor de '_varPref', ordenado por preferencia del
        // Agente, chequeamos si produce que otro Agente se quede sin
        // asignar:
        // - Si la respuesta es NO, seleccionamos este valor
        // - En otro caso, continuamos
        // Si después de intentarlo con todos los valores, todos ellos
        // dejan algún Agente sin asignar, elegimos el más preferido.

        return IlcOr ( _varPref == valPref,
                       IlcAnd(_varPref != valPref, this), label );
    }
}

```

- *InstanciaVariableEnumeradoConjuntos*

```

IlcGoal InstanciaVariableInferidoEnumeradoConjuntosI::execute() {
    IlcIntArray conjPref;

    // Chequeamos si podemos hacer un backtrack inteligente, ya sea
    // por una preferencia o por un NO_ASIGNADO. En caso de poder
    // hacerlo, se salta desde aquí, así que la ejecución de este
    // Goal se interrumpe.

    if ( !_varPref.isBound() ) {
        // Para cada valor de la lista de conjuntos, tal que interseca con
        // el dominio de '_varPref', ordenado por preferencia del
        // Agente, chequeamos si produce que otro Agente se quede sin
        // asignar:
        // - Si la respuesta es NO, seleccionamos este valor
        // - En otro caso, continuamos
        // Si después de intentarlo con todos los valores, todos ellos
        // dejan algún Agente sin asignar, elegimos el más preferido.

        return IlcOr ( IlcMember(_varPref, conjPref),
                       IlcAnd(IlcNotMember(_varPref, conjPref),
                              this) ) );
    }
}

```