**Reporte: Problema de Construcción de Casas con Recursos Unarios**

**Descripción del Problema**

Este problema consiste en **programar la construcción de 5 casas** utilizando **2 trabajadores** (Jim y Joe), donde cada casa requiere las mismas 10 tareas, pero deben ser programadas considerando múltiples restricciones y costos.

**Características principales:**

* **5 casas** para construir en diferentes ubicaciones
* **2 trabajadores** especializados en diferentes tareas
* **10 tareas** por casa con duraciones específicas
* **Tiempos de transición** entre casas para los trabajadores
* **Costos por tardanza** y duración total

**Variables de Decisión**

**1. Variables de Intervalo para Casas**

dvar interval houses [h in Houses] in ReleaseDate[h]..(maxint div 2)-1;

Representa el intervalo total de construcción de cada casa, desde el inicio de la primera tarea hasta el final de la última tarea.

**2. Variables de Intervalo para Tareas**

dvar interval itvs[h in Houses][t in TaskNames] size Duration[t];

Matriz bidimensional que representa cada tarea de cada casa con su duración específica.

**3. Variables de Secuencia para Trabajadores**

dvar sequence workers [w in WorkerNames] in

all(h in Houses, t in TaskNames: Worker[t]==w) itvs[h][t] types

all(h in Houses, t in TaskNames: Worker[t]==w) h;

Define el orden en que cada trabajador realiza sus tareas asignadas, incluyendo información de ubicación (casa) para calcular tiempos de transición.

**4. Tiempos de Transición**

{triplet} transitionTimes = { <i,j, ftoi(abs(i-j))> | i in Houses, j in Houses };

Define el tiempo que tarda un trabajador en moverse de la casa i a la casa j, basado en la distancia absoluta entre ubicaciones.

**Restricciones**

**1. Restricciones de Precedencia**

forall(h in Houses)

forall(p in Precedences)

endBeforeStart(itvs[h][p.pre], itvs[h][p.post]);

Asegura que las tareas precedentes terminen antes de que comiencen las tareas siguientes **para cada casa**.

**2. Restricciones de Cobertura (Span)**

forall(h in Houses)

span(houses[h], all(t in TaskNames) itvs[h][t]);

La variable de intervalo de cada casa debe cubrir exactamente todas sus tareas, comenzando con la primera tarea y terminando con la última.

**3. Restricciones de No Solapamiento**

forall(w in WorkerNames)

noOverlap(workers[w], transitionTimes);

Cada trabajador solo puede realizar una tarea a la vez, y debe respetar los tiempos de transición entre casas.

**Función Objetivo**

minimize sum(h in Houses)

(Weight[h] \* maxl(0, endOf(houses[h])-DueDate[h]) + lengthOf(houses[h]));

El objetivo es **minimizar el costo total** que incluye:

1. **Costo por tardanza**: Weight[h] \* maxl(0, endOf(houses[h])-DueDate[h])
   * Penaliza cada día que la casa se complete después de su fecha de vencimiento
   * La casa 3 tiene mayor peso (200.0) comparado con las otras (100.0)
2. **Costo por duración**: lengthOf(houses[h])
   * Penaliza el tiempo total que toma construir cada casa
   * Incentiva completar las casas más rápidamente

**Aspectos Técnicos Importantes**

**Recursos Unarios**

Los trabajadores son **recursos unarios**, significa que cada uno puede realizar solo una tarea a la vez. Esto se modela mediante:

* Variables de secuencia que definen el orden de tareas
* Restricciones noOverlap que evitan solapamiento temporal

**Tiempos de Transición**

Cuando un trabajador termina una tarea en una casa y debe realizar la siguiente en otra casa diferente, debe invertir tiempo en transportarse. Esto se modela con:

* Una tupla que define tiempos entre ubicaciones
* La distancia se calcula como abs(i-j) entre casas

**Optimización con Límites**

execute {

cp.param.FailLimit = 20000;

}

Se establece un límite de fallas para evitar tiempos de ejecución excesivos, permitiendo obtener una buena solución, aunque no necesariamente óptima.

**Resultados Esperados**

El modelo debe generar una programación que:

1. Respete todas las precedencias de tareas
2. Asigne trabajadores sin solapamientos
3. Considere tiempos de transición entre casas
4. Minimice costos por tardanza y duración
5. Muestre la secuencia óptima en un diagrama de Gantt

**Conclusión**

Este problema demuestra la complejidad de la programación de proyectos con recursos limitados, donde además de las restricciones temporales tradicionales, se deben considerar aspectos logísticos como los tiempos de transición y múltiples objetivos de costo. La solución requiere un balance cuidadoso entre minimizar tardanzas y minimizar la duración total del proyecto.