Proyecto Inteligencia Artificial

Estado de Avance

Ian Rossi A.

Universidad Técnica Federico Santa María
Departamento de Informática

June 9, 2024

Contenido

1 Representación y Manejo de Restricciones

2 Algoritmo

3 Conclusiones

Contenido

1 Representación y Manejo de Restricciones

2 Algoritmo

3 Conclusiones

Tupla representativa

Como se indicó en el material de referencia de la *Société française de Recherche Opérationnelle et Aide à la Décision (ROADEF) 2005* (Christine Solnon et al. 2008).

(V, O, p, q, r), donde:

- $V = v_1, v_2, ..., v_n$ es el conjunto de clases a ser producidos.
- $O = o_1, o_2, ..., o_m$ es el conjunto de opciones que pueden ser elegidas para cada clase.
- $p_i : O \rightarrow \mathbb{N}$ y $q_i : O \rightarrow \mathbb{N}$ son la capacidad relativa y absoluta respectivamente.
- $r_{ij} : V \times O \rightarrow 0, 1$ corresponde a la matriz que indica si una opción o_i debe ser instalada en un vehículo v_j.

Restricciones

Recordemos las restricciones.

- $\sum_{i \in O} \sum_{s \in X} \sum_{j \in s} r_{ij} - p_i > 0, \forall o_i \in O$

Restricciones

Recordemos las restricciones.

- $\sum_{i \in O} \sum_{s \in X} \sum_{j \in s} r_{ij} - p_i > 0, \forall o_i \in O$
- Establece que no existan subsecuencias s de tamaño q_i que sobrepasen a p_i para toda opción.

Restricciones

Recordemos las restricciones.

- $d_j = \sum_{j \in V} \sum_{i \in O} r_{ij}, \forall j \in V$

Restricciones

UNIVERSIDAD TECNICA FEDERICO SANTA MARIA
DEPARTAMENTO DE INFORMÁTICA

Recordemos las restricciones.

- $d_j = \sum_{j \in V} \sum_{i \in O} r_{ij}, \forall j \in V$
- La demanda de cada clase se debe cumplir siempre

Es relativamente simple establecer ambas restricciones en el código

- Restricción 1: Dentro de la matriz r, se encuentra la *row* que corresponda a la opción o_i, y se suman los 1 largo de esta, restando el p_i correspondiente y revisando que sea mayor a 0.

Restricciones

Es relativamente simple establecer ambas restricciones en el código

- Restricción 1: Dentro de la matriz r, se encuentra la *row* que corresponda a la opción o_i, y se suman los 1 largo de esta, restando el p_i correspondiente y revisando que sea mayor a 0.
- Esta restricción es la base de la función de evaluación

Restricciones

Es relativamente simple establecer ambas restricciones en el código

- Restricción 2:

Restricciones

Es relativamente simple establecer ambas restricciones en el código

- Restricción 2:

```
bool check_demand(vector<int> x) {
    for (int curr_class = 0; curr_class < instance.r.size(); curr_class
     ++) {
        if (instance.d[curr_class] > accumulate(x.begin(), x.end(), 0))
     {
            return false;
        }
    }
}
```

Representación

Dado que se sabe el tamaño de la secuencia completa de autos X mediante los parámetros de inicialización, podemos representar la solución como un vector de una dimensión `int x[num_cars];`.

```
vector<int> x[6] = {4, 2, 3, 1, 6, 5}
```

Ejemplo de una instancia posible para X

Contenido

1 Representación y Manejo de Restricciones

2 Algoritmo

3 Conclusiones

Solución inicial

Primero se deben determinar los parámetros iniciales del problema (V, O, p, q, r). Esto se realiza asignando valores aleatorios a una serie de variables, con algunas restricciones, o leyendolos desde una instancia predeterminada:

Solución inicial

Primero se deben determinar los parámetros iniciales del problema (V, O, p, q, r). Esto se realiza asignando valores aleatorios a una serie de variables, con algunas restricciones, o leyendolos desde una instancia predeterminada:

- `num_cars` El número de autos en la secuencia X, dígase el largo de la solución

UNIVERSIDAD TECNICA FEDERICO SANTA MARIA
DEPARTAMENTO DE INFORMÁTICA

Primero se deben determinar los parámetros iniciales del problema (V, O, p, q, r). Esto se realiza asignando valores aleatorios a una serie de variables, con algunas restricciones, o leyendolos desde una instancia predeterminada:

- `num_cars` El número de autos en la secuencia X, dígase el largo de la solución
- `num_options` El número de opciones

Solución inicial

Primero se deben determinar los parámetros iniciales del problema (V, O, p, q, r). Esto se realiza asignando valores aleatorios a una serie de variables, con algunas restricciones, o leyendolos desde una instancia predeterminada:

- `num_cars` El número de autos en la secuencia X, dígase el largo de la solución
- `num_options` El número de opciones
- `num_classes` El número de clases

Solución inicial

Luego tenemos que establecer p_i y q_i. Esto se puede hacer hacer mediante `std::vector` de 1 dimensión.

Restricción

La parte importante de este punto, es asegurarse de que para todo o_i, p_i no sea mayor a q_i, de lo contrario se entra en incoherencias según el modelo matemático.

$$p_i \leq q_i$$

Solución inicial

Finalmente tenemos que establecer r_{ij}. Al igual que en los puntos anteriores, el tamaño es predefinido, así que podemos usar un `std::vector` de 2 dimensiones:

```
rvector<vector<int» r(num_classes, vector<int> (num_options, 0));
```

```cpp
vector<vector<int>> r = {
{1, 0, 1, 1, 0},
{0, 0, 0, 1, 0},
{0, 1, 0, 0, 1},
{0, 1, 0, 1, 0},
{1, 0, 1, 0, 0},
{1, 1, 0, 0, 0},
};
```

Ejemplo de una instancia posible para r

Solución inicial

Con todos estos parámetros establecidos podemos crear la solución inicial o punto de partida:

```
vector<int> x (V, 1)
```

`Vector X de largo V inicializado en 0`

Función de evaluación

La función de evaluación se establece como el inverso de la cantidad de violaciones a las restricciones. Esto representa la "fluidez" de la línea de ensamblaje.

```
int eval(vector<int> x) {
    int sum = 0;
    for (int option = 0; option < instance.O; option++) {
        int curr_p = instance.p[option];
        int curr_q = instance.q[option];
        vector<int> x_bin = get_classes(x, option);
        for (int i = 0; i < x_bin.size(); i++) {
            vector<int> sub(&x_bin[i],&x_bin[i + curr_q]);
            sum = sum + accumulate(sub.begin(), sub.end(), 0) - curr_p;
        }
    }
    return sum;
}
```

Movimiento

Teniendo en cuenta que X es un conjunto donde X_i corresponde a la clase del i-ésimo vehículo en la secuencia.

Podemos establecer el movimiento como tomar un X_i y cambiarlo a la clase siguiente en el conjunto que las contiene, y de ser el último, volver al primero.

$$X_i = 2 \rightarrow X_i = 3$$

Este iría iterando sobre X, de manera de revisar todos los elementos de este.

```
vector<int> move(vector<int> x, int iteration) {
    int pos = iteration % x.size();
    pos--;  // esto para que la numero de iteracion corresponda al elemento
     del vector x
    x[pos] = x[pos] + 1;
    return x;
}
```

Algoritmo Greedy

El algoritmo greedy en cada una de sus iteraciones realiza un movimiento si es que mejora la función de evaluación o no hay cambio en esta, de lo contrario sigue adelante con el siguiente elemento de X.

En el caso de que termine de recorrer el conjunto X, vuelve a empezar. Se detiene cuando llega al número de iteraciones determinado.

```
vector<int> greedy(int iterations, problem instance, vector<int> x) {
    int iteration = 0;
    int curr_fitness = eval(x);
    for (int i = 0; i < iterations; i++) {
        vector<int> next_x = move(x, iteration);
        if (eval(next_x) < curr_fitness) { // realiza el movimiento si mejora la fitness
            x = next_x;
            curr_fitness = eval(x);
        }
        iteration++; // incrementa la iteracion si o si
    }
    return x;
}
```

Plan para SA

Utilizar un algoritmo del tipo *Simulated Annealing* como se vió en clases. Realizar un movimiento si es que incurre en una mejora, y en caso de que el movimiento cree una solución peor, someterlo a $P(.) = \frac{\Delta e}{T}$

Los detalles como la temperatura inicial, el decrecimiento de esta, y la existencia de fases de recalentamiento serán evaluadas al momento de implementarse.

Contenido

1 Representación y Manejo de Restricciones

2 Algoritmo

3 Conclusiones

Conclusiones

Durante la implementación de los puntos anteriormente hablados resaltaron los siguientes problemas:

- La función objetivo detallada en el estado del arte es incorrecta.

Conclusiones

Durante la implementación de los puntos anteriormente hablados resaltaron los siguientes problemas:

- La función objetivo detallada en el estado del arte es incorrecta.
- La restricción 1 detallada en el estado del arte es incorrecta.

Conclusiones

Durante la implementación de los puntos anteriormente hablados resaltaron los siguientes problemas:

- La función objetivo detallada en el estado del arte es incorrecta.
- La restricción 1 detallada en el estado del arte es incorrecta.
- La lectura de una instancia pre-cosntruida es mas complejo de lo esperado.

Conclusiones

UNIVERSIDAD TECNICA FEDERICO SANTA MARIA
DEPARTAMENTO DE INFORMÁTICA

Durante la implementación de los puntos anteriormente hablados resaltaron los siguientes problemas:

- La función objetivo detallada en el estado del arte es incorrecta.
- La restricción 1 detallada en el estado del arte es incorrecta.
- La lectura de una instancia pre-cosntruida es mas complejo de lo esperado.
- Dado una instanciación del problema según la tupla (V, O, p, q, r) no es trivial el determinar si es que existe una solución perfecta, es decir, que no incumpla ninguna de las restricciones (CSPLib).