Estado de Avance ESSP

Gabriel Ortiz Ángel

Representación

Secuencia de n*h que represente el turno CT que escoge el trabajador i en el día d.

Ejemplo: “1321101012”

n = 5;
h = 2;
CT = 3;

Día 1, 5 trabajadores,
3 tipos de turnos.

*0 es día sin turno.

Representación en código

Vector 2D de enteros apoyado por un vector 3D booleano para restricciones*.

- $x_{i,d}$: Turno asignado al empleado i en el día d.
- $y_{i,d,t}$: Si el turno t fue asignado a i en d.

*Representación basada en Dahmen (1) y Carter (2).

Lectura de Instancia

1. Variables unitarias. → Variable primitiva

2. Variables separadas por ‘,’ → Vector

3. Variables separadas por ‘|’ → Map

*Los maps luego de la lectura fueron traducidos a vectores.

Ejemplo

```
SECTION_HORIZON
# All instances start on a Monday
# The horizon length in days:
28

SECTION_SHIFTS
# ShiftID, Length in mins, Shifts which cannot follow this shift | separated
E,480,
D,480,E
L,480,E|D
```

Primitiva

Vector

Map

Ejemplo

```
A,E=28|D=28|L=0,8640,7680,5,2,2,2
B,E=0|D=0|L=28,8640,7680,5,2,2,2
C,E=0|D=28|L=28,8640,7680,5,2,2,2
D,E=28|D=28|L=0,8640,7680,5,2,2,2
```

Primitiva

Vector

Map

Obtención de Solución Inicial

- **Punto de Inicio:** Se empieza desde una solución parcial básica, por ejemplo la secuencia “1”.

- **Funcion Miope:** Se **prueban** distintos **valores** para el último número agregado, evaluando la solución **completa**. Se escoge la solución que esté mejor evaluada, y se repite el ciclo **agregando otro “1”**.

```cpp
void solve_greedy(vector<vector<int>>& sol, vector<vector<vector<bool>>>& sol_bit){
    // Checks turns per worker per day, and chooses the one that minimizes the function.
    for (int i = 0; i < n; i++){
        for (int d = 0; d < h; d++){
            int original_t = sol[i][d];
            int min_eval = eval_sol(sol, sol_bit);
            int min_turn = original_t;

            for (int t = 0; t < CT; t++){
                if (t == original_t) continue;

                int prev_shift = sol[i][d];
                sol[i][d] = t;
Auxiliar        set_y_from_x(sol_bit, t, i, d, prev_shift);
                int new_eval = eval_sol(sol, sol_bit);
                int is_min = new_eval <= min_eval;
                min_eval = is_min? new_eval : min_eval;
                min_turn = is_min? t : min_turn;
            }
```

Turno original

Sol actual es el mínimo

Revisar calidad con otros turnos

```
            // cout << min_eval << endl;
            int prev_shift = sol[i][d];
            sol[i][d] = min_turn;
            set_y_from_x(sol_bit, min_turn, i, d, prev_shift);

            if (d < h - 1){
                sol[i].push_back(1);
                sol_bit[i].push_back(vector<bool>(CT, false));
                sol_bit[i][d+1][1] = true;
            }
        }

        if (i < n - 1){
            sol.push_back(vector<int>(1, 1));
            sol_bit.push_back(vector<vector<bool>>(1, vector<bool>(CT, false)));
            sol_bit[i + 1][0][1] = true;
        }
    }
    return;
```

Asignar el mejor

Agregar el siguiente día a la solución

Agregar nuevo empleado a la solución

Restricciones

1. Restricciones blandas → Penalización **definida** en instancia
2. Restricciones duras → Penalización **enorme**

Función de Evaluación

- Penalizaciones **fijas** por romper retricciones binarias.
  - Días libres obligatorios.
- Penalizaciones que **aumentan** según la **distancia** respecto al valor ideal.
  - Máximo y mínimo de minutos a trabajar.

Función de Evaluación: Código

```
int eval_sol(vector<vector<int>> sol, vector<vector<vector<bool>>> sol_bit){
    // Returns the total penalty for the solution
    int staff = sol.size();
    int days = sol[0].size();
    day_turn_qty.clear();
    day_turn_qty.resize(days, vector<int>(CT, 0));
    penalty_day_turn.clear();
    penalty_day_turn.resize(days, vector<int>(CT, 0));
    penalty_per_worker.clear();
    penalty_per_worker.resize(staff, 0);
    broken_h_constr = 0;
    int hard_penalty = eval_hard_constraints(sol, sol_bit);
    int soft_penalty = eval_soft_constraints(sol, sol_bit);
    return hard_penalty + soft_penalty;
}
```

Reiniciar variables

Suma de penalizaciones

Evaluación Restricciones Duras

```
int eval_hard_constraints(vector<vector<int>> sol, vector<vector<vector<bool>>> sol_bit){
    // evaluate hard constraints, adding BIG penalty for each one.
    int staff = sol.size();
    int penalty = 0;
    vector<int> total_min (staff, 0);
    vector<vector<int>> person_turn_qty(staff, vector<int>(CT, 0));
    bool is_broken_1, is_broken_2;
```

Empleados en la solucion

Variable de retorno

Estructuras auxiliares

```
for (int i = 0; i < staff; i++){
    int curr_days = sol[i].size(); Días considerados para empleado
    for (int t = 0; t < CT; t++){
        for (int d = 0; d < curr_days; d++){
            if (t == 0){
                is_broken_1 = ((!(sol_bit[i][d][t]) && LO_i_d[i][d]) == 1);
                // if (is_broken_1) cout << "Restriccion dias libres obligatorios rota" << endl;
                penalty += is_broken_1 * PENALTY_COST; // Mandatory days-off (13)
                broken_h_constr += is_broken_1;
            }
            else{
                if (d != 0){
                    int previous_day_turn = sol[i][d - 1];
                    is_broken_1 = ((R_t_k[t][previous_day_turn] && sol_bit[i][d][t]) == 1);
                    // if (is_broken_1) cout << "Restriccion turnos consecutivos rota" << endl;
                    penalty += is_broken_1 * PENALTY_COST; // Consecutive Turns (3)
                    broken_h_constr += is_broken_1;
                }
                person_turn_qty[i][t] += sol_bit[i][d][t]; // Turns per person per shift (4)
                total_min[i] += sol_bit[i][d][t] * L_t[t - 1]; // Minutes per person (5)
            }
        }
    }
}
```

Sección diferente: Días consecutivos

```
for (int i = 0; i < staff; i++){
    int curr_days = sol[i].size();
    int work_streak = 0;
    int off_streak = 0;
    for (int d = 0; d < curr_days; d++){
        if (sol[i][d] != 0 && curr_days >= DL_i[i]){
            work_streak += 1;
            if (curr_days < h){ // Needed to help greedy solution
                if (d % 7 == 5) penalty += PENALTY_COST;
                int max_broken = work_streak > CMA_i[i];
                // if (max_broken) cout << "Restriccion maximo turnos consecutivos rota" << endl;
                penalty += max_broken? PENALTY_COST: 0;
                broken_h_constr += max_broken;
            }

            is_broken_1 = (off_streak != 0 && off_streak < DL_i[i]);
            // if (is_broken_1) cout << "Restriccion minimo de días libres consecutivos rota" << endl;
            penalty += is_broken_1 * PENALTY_COST; // (9 - 10)
            broken_h_constr += is_broken_1;
            off_streak = 0;
        }
```

Overstaff y Understaff

```
for (int i = 0; i < staff; i++){
    int curr_days = sol[i].size();
    for (int t = 1; t < CT; t++){
        for (int d = 0; d < curr_days; d++){
            day_turn_qty[d][t] += sol_bit[i][d][t]; // Over Staff and Under Staff (14 - 17)

            penalty_get = sol_bit[i][d][t] * PAT_i_d_t[i][d][t];
            penalty += penalty_get;
            penalty_per_worker[i] += penalty_get != 0;

            penalty_get = (sol_bit[i][d][t] == 0) * PNAT_i_d_t[i][d][t];
            penalty += penalty_get;
            penalty_per_worker[i] += penalty_get != 0;
        }
    }
}
```

Tabu Search

- Propuesto por Fred Glover en 1986. (3).
- Recorre y evalúa vecindarios de una solución actual.
- Escoge la solución mejor evaluada.
- El movimiento de esa solución es agregado a una **lista tabú** para que no se repita.

Movimiento

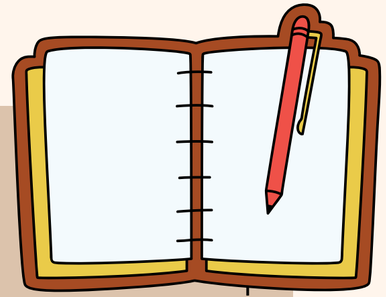
Cambiar un turno al siguiente*.

Ejemplo: “13211” -> “10211”

- Sencillo de implementar, reduce el vecindario a **n*h** y considera todos los turnos.

*Uso de variables para revisar si un turno es posible.

Conclusiones



- Falta de **optimización** para evaluación, sobre todo para TS.
  - Revisar formas de **evaluar** solo **cambios**.
- Solución inicial no logra **ver el futuro.**
  - Usar variables para **predecir.**

Referencias

1. Sana Dahmen and Monia Rekik. Solving multi-activity multi-day shift scheduling problems with a hybrid heuristic. Journal of Scheduling, 18(2):207-223, 2014.}
2. Michael W. Carter and Sophie D. Lapierre. Scheduling emergency room physicians. Health Care Management Science, 4(4):347-360, 2001.
3. Fred Glover. Future paths for integer programming and links to arti□cial intelligence. Computers & Operations Research, 13(5):533-549, January 1986.