SISTEM DE PROGRAMARE (TIMETABLING)

Proiect cu algoritm Forward Checking

---

1. Introducere

În diverse organizații, echipe sau grupuri de lucru, gestionarea programelor pentru o eventuală întâlnire (meeting) poate deveni un demers anevoios. Fiecare persoană își poate nota disponibilitatea în intervale orare diferite, uneori pe mai multe zile. Problema devine și mai complexă pe măsură ce crește numărul participanților.

Proiectul de față își propune să adreseze această provocare, dezvoltând o **aplicație de tip timetabling** (sistem de programare) care:

- Permite **introducerea** și **gestionarea** unei liste de utilizatori (în mod dinamic).
- Stochează **intervalele orare** disponibile pentru fiecare utilizator, asociate diverselor zile.
- Folosește un **algoritm CSP** (Constraint Satisfaction Problem) cu **Forward Checking**, pentru a determina **intervalul comun** de disponibilitate pentru toți utilizatorii.

În continuare, vom descrie **problema** abordată, **algoritmul** utilizat, **modul de implementare** și **rezultatele obținute**.

2. Descrierea problemei

Problema principală se poate rezuma astfel:
„Dându-se nn utilizatori, fiecare cu un set de zile și intervale orare (ex. HH:MM-HH:MM), să se găsească porțiunile orare (dacă există) în care **toți** utilizatorii sunt simultan disponibili, pentru fiecare zi solicitată.”

Pentru a simplifica, fiecare utilizator are un set de intervale (start, end) pe fiecare zi, exprimate cu rezoluție de minute. De pildă, utilizatorul *Alice* poate fi disponibil în data de 09.01.2025 între 07:00 și 09:20, respectiv între 10:00 și 12:00. La rândul său, utilizatorul *Bob* poate fi disponibil la aceeași dată între 08:00 și 09:15. Intersecția comună va fi 08:00–09:15.

Problema devine mai complicată pe măsură ce crește numărul utilizatorilor și zilele implicate. Un algoritm simplu de intersectare „brut” (pairwise intersection) poate fi utilizat cu succes, însă, în cazul integrării într-un **CSP**, avem beneficii suplimentare, precum posibilitatea **extinderii** la alte tipuri de constrângeri (de exemplu, să existe pauză minimă de 15 minute, să se ocupe doar anumite sloturi etc.).

3. Aspecte teoretice privind algoritmul CSP și Forward Checking

3.1. Modelarea problemei ca un CSP

Un **Constraint Satisfaction Problem** (CSP) este definit de:

- **Variabile**: setul de entități care trebuie să fie atribuite unei valori.
- **Domenii**: pentru fiecare variabilă, un set de valori posibile pe care le poate lua.
- **Constrângeri**: relații care specifică ce combinații de valori sunt permise.

În cazul nostru:

- **Variabilele** pot fi asimilate cu *utilizatorii* (și implicit, pentru fiecare utilizator, trebuie să alegem un interval orar valabil).
- **Domeniile** reprezintă **mulțimea de intervale** (TimeSpan Start, TimeSpan End) disponibile pentru fiecare utilizator, pentru o anumită zi.
- **Constrângerea** fundamentală este că trebuie să existe **intersecție de timp** între intervalul ales pentru un utilizator și intervalele alese pentru ceilalți, astfel încât la final să existe un bloc de timp comun.

3.2. Principiul algoritmului Forward Checking

„*Forward Checking*" este o metodă de **backtracking** îmbunătățită, utilizată de regulă în rezolvarea CSP-urilor, cu scopul de a **evita explorarea** unor căi care în mod evident vor conduce la eșec.

Pașii algoritmului:

1. **Selectarea variabilei** (în proiectul nostru, alegem un utilizator). Se parcurge domeniul acestuia (lista de intervale) și se încearcă, pe rând, fiecare interval.

2. **Compatibilitatea** se verifică prin filtrarea domeniilor celorlalți utilizatori (variabile neasignate). Mai exact, la fiecare pas:

    - Se alege un interval orar *I* pentru utilizatorul curent.
    - Se **reduce** domeniul utilizatorilor încă nealocați, **păstrând** doar acele intervale care se **suprapun** cu *I*.
    - Dacă pentru oricare dintre utilizatorii următori rămâne un **domeniu gol**, **facem backtrack** (ne întoarcem și alegem alt interval pentru utilizatorul curent).

3. **Recursivitate**: După reducerea domeniilor, trecem la **următorul utilizator**. Dacă reușim să alocăm un interval pentru fiecare utilizator, atunci avem o **soluție**.

3.3. Operația „Reduce Domains”

„Reduce Domains” (pasul esențial în Forward Checking) menține doar valorile care **nu încalcă** constrângerile determinate de intervalul ales până la momentul respectiv.

Formal, pentru un interval ($Start_{current}$, $End_{current}$) al utilizatorului curent, oricare interval (Start viitor,End viitor)(Start_{viitor}, End_{viitor}) din domeniul utilizatorului viitor (next user) rămâne valabil doar dacă cele două intervale se intersectează:

$$Overlap \Leftrightarrow Start_{current} < End_{viitor} \ ^{\wedge} \ Start_{viitor} < End_{current}$$

Prin eliminarea din domeniu a tuturor intervalelor care nu se suprapun, se asigură că, la pasul următor, nu vom încerca alocări imposibile.

4. Modalitatea de rezolvare

4.1. Structura aplicației

- **Interfață grafică (Windows Forms)**

    1. Un ComboBox pentru lista de utilizatori.
    2. Un MonthCalendar pentru selectarea zilelor.
    3. Un TextBox (hoursTextBox) unde se introduce intervalul orar în format HH:MM-HH:MM.
    4. Un buton „Add Schedule” care salvează intervalul pentru utilizatorul selectat, în ziua calendaristică indicată.
    5. Un buton „Add User” care permite adăugarea dinamică a unui nou utilizator (prin introducerea numelui).
    6. Un buton „Load Schedule” care încarcă datele dintr-un fișier JSON (pentru a nu reintroduce manual).
    7. Un buton „Run Algorithm” care execută procedura de forward checking pentru toate zilele selectate în calendar.

Structura de date:

$$userAvailability[userName][date] = List < (TimeSpan\ Start,\ TimeSpan\ End) >$$

- Astfel, pentru fiecare userName (cheie de tip string), avem un dicționar care mapează o dată (DateTime) la o listă de intervale (Start, End).

- **Algoritmul**:

    1. Se identifică toți utilizatorii din ComboBox.

    2. Se parcurge intervalul de zile din MonthCalendar.SelectionRange.Start până la SelectionRange.End.

    3. Pentru fiecare zi, se extrage domeniul fiecărui utilizator (dacă nu există intervale, domeniul e gol).

    4. Se rulează **Forward Checking** pentru a găsi un interval comun (sau a constata lipsa acestuia).

5. Implementarea algoritmului Forward Checking (cu Reduce Domains)

Mai jos evidențiem secvențele de cod relevante, cu explicații:

5.1. Apelul algoritmului

```
private void btnRunAlgorithm_Click(object sender, EventArgs e)
{
    // Se curăță listSchedules pentru a afișa noile rezultate
    listSchedules.Items.Clear();

    // Luăm toți utilizatorii
    var allUsers = boxUsers.Items.Cast<string>().ToList();
```

```
// Intervalul de date selectate în calendar
DateTime day = calendar.SelectionRange.Start;
DateTime lastDay = calendar.SelectionRange.End;

// Iterăm peste fiecare zi
while (day <= lastDay)
{
    // Construim domeniul (lista de intervale) pentru fiecare utilizator
    var domains = new Dictionary<string, List<(TimeSpan, TimeSpan)>>();
    foreach (var user in allUsers)
    {
        if (userAvailability.ContainsKey(user) && userAvailability[user].ContainsKey(day))
            domains[user] = new List<(TimeSpan, TimeSpan)>(userAvailability[user][day]);
        else
            domains[user] = new List<(TimeSpan, TimeSpan)>();
    }

    // Apelăm Forward Checking
    var result = ForwardChecking(allUsers, domains, 0, null);

    // Interpretăm rezultatul
    if (result == null || result.Count == 0)
```

```
        {
            listSchedules.Items.Add($"{day.ToShortDateString()} => No common intersection");
        }
        else
        {
            listSchedules.Items.Add($"{day.ToShortDateString()} => Intersection(s):");
            foreach (var (s, e2) in result)
            {
                listSchedules.Items.Add($"    {s:hh\\:mm}-{e2:hh\\:mm}");
            }
        }

        day = day.AddDays(1);
    }
}
```

Explicații:

- ForwardChecking ne va returna fie o listă de intervale (intersecția finală), fie null dacă nu s-a putut găsi nicio soluție.
- Se reia procedeul pentru fiecare zi din intervalul selectat din calendar.

5.2. Metoda ForwardChecking

```
private List<(TimeSpan, TimeSpan)> ForwardChecking(
    List<string> users,
    Dictionary<string, List<(TimeSpan, TimeSpan)>> domains,
    int userIndex,
    List<(TimeSpan, TimeSpan)> currentIntersection)
{
    // Dacă am atribuit intervale tuturor userilor, returnăm intersecția finală
    if (userIndex == users.Count)
        return currentIntersection;

    // Alegem userul curent
    var user = users[userIndex];
    var userDomain = domains[user];

    // Dacă acest user nu are niciun interval valabil, eșec
    if (userDomain.Count == 0)
        return null;

    // Încercăm fiecare interval din domeniul userului curent
    foreach (var interval in userDomain)
    {
        // Calculăm noua intersecție
        List<(TimeSpan, TimeSpan)> newIntersection;
        if (currentIntersection == null)
```

```
        {
            // Primul user - intersecția e doar intervalul ales
            newIntersection = new List<(TimeSpan, TimeSpan)> { interval };
        }
        else
        {
            // Intersectăm intervalul cu intersecția actuală
            newIntersection = IntersectListWithInterval(currentIntersection, interval);
            if (newIntersection.Count == 0)
            {
                // Fără suprapunere, trecem mai departe
                continue;
            }
        }
        // Aplicăm forward checking: reducerea domeniilor userilor următori
        var reducedDomains = ReduceDomains(users, domains, userIndex, interval);
        // Apel recursiv
        var solution = ForwardChecking(users, reducedDomains, userIndex + 1,
newIntersection);
        // Dacă am găsit o soluție validă, o returnăm
        if (solution != null && solution.Count > 0)
            return solution;
    }

    // Nu există nicio atribuire validă pentru userul curent
```

```
    return null;
}
```

Comentariu:

- currentIntersection păstrează intersecția de intervale găsită până la acest moment. Dacă e null, înseamnă că suntem la primul utilizator și orice interval poate fi baza intersecției.
- „newIntersection = IntersectListWithInterval(...)” calculează intersecția dintre currentIntersection (deja filtrată de userii anteriori) și intervalul curent.

5.3. Metoda ReduceDomains (cheia Forward Checking)

```
private Dictionary<string, List<(TimeSpan, TimeSpan)>> ReduceDomains(
    List<string> users,
    Dictionary<string, List<(TimeSpan, TimeSpan)>> domains,
    int userIndex,
    (TimeSpan Start, TimeSpan End) assignedInterval)
{
    // Facem o copie a domeniilor pentru a nu distruge originalul
    var newDomains = new Dictionary<string, List<(TimeSpan, TimeSpan)>>();
    foreach (var kvp in domains)
    {
        newDomains[kvp.Key] = new List<(TimeSpan, TimeSpan)>(kvp.Value);
    }
```

```
    // Reducem domeniile pentru utilizatorii următori
    for (int i = userIndex + 1; i < users.Count; i++)
    {
        var futureUser = users[i];
        var originalList = newDomains[futureUser];
        var filtered = new List<(TimeSpan, TimeSpan)>();

        // Păstrăm doar intervalele ce se suprapun cu assignedInterval
        foreach (var candidate in originalList)
        {
            if (IntervalsOverlap(candidate, assignedInterval))
            {
                filtered.Add(candidate);
            }
        }

        newDomains[futureUser] = filtered;
    }

    return newDomains;
}
```

Explicații:

- Metoda primește assignedInterval, care este intervalul deja ales pentru utilizatorul curent.

- Pentru fiecare utilizator **viitor** din listă, se filtrează lista de intervale, astfel încât să rămână doar cele care se **suprapun** cu intervalul ales.

- Dacă la final, pentru vreun utilizator, filtered devine gol, la pasul următor se va constata un eșec și se va face **backtrack**.

6. Rezultate obținute și exemple de rulare

6.1. Exemplu cu două zile selectate

- **Utilizatori**: Alice, Bob și Carol.
- **Date**: 09.01.2025 și 10.01.2025.

Ziua 09.01.2025:

- Alice: [07:00–09:20], [10:00–12:00]
- Bob: [08:00–09:15]
- Carol: [08:30–09:00], [10:00–11:00]

Aplicația identifică:

- Intersecția [08:00–09:15] cu [08:30–09:00] dă [08:30–09:00] (comun Bob & Carol).
- Comparată cu Alice ([07:00–09:20]), rămâne [08:30–09:00].

Rezultatul final: **08:30–09:00**.

Ziua 10.01.2025:

- Alice: [07:30–09:40]
- Bob: [07:50–09:20]
- Carol: [07:45–08:50]

Intersecția comună 07:50–08:50 (Bob & Carol) intersectată cu 07:30–09:40 (Alice) = **07:50–08:50**.

Capturi de ecran pot evidenția conținutul din ListBox:

```
09.01.2025 => Intersection(s):
   08:30-09:00
10.01.2025 => Intersection(s):
   07:50-08:50
```

6.2. Caz în care nu există intersecție

Dacă la o anumită dată, un utilizator are intervale total necompatibile cu ceilalți (de exemplu, 14:00–16:00 versus 08:00–10:00 la ceilalți), aplicația va afișa:
No common intersection pentru ziua respectivă.

7. Concluzii

Aplicația realizată oferă un mod **intuitiv** de a defini disponibilitățile utilizatorilor și de a **calcula automat** intervalul comun, folosind algoritmul CSP (Forward Checking + Reduce Domains). Avantajele acestei metode includ:

1. **Reutilizare**: Modelul CSP permite extinderea la alte tipuri de constrângeri (de ex., pauze, restricții de date etc.).
2. **Eficiență sporită** față de backtracking-ul simplu, deoarece prin „forward checking" eliminăm din timp variantele imposibile.
3. **Flexibilitate**: Putem adăuga oricâți utilizatori și oricâte intervale, inclusiv pe multiple zile consecutive, fără să schimbăm mult logica.

Eventualele **limitări** sau posibile îmbunătățiri sunt:

- Integrarea unui modul de **salvare** a modificărilor în fișier (pentru a se putea relua ulterior).

- Oferirea unei **vizualizări grafice** a intervalelor pe un timeline, pentru o mai bună lizibilitate.

- Posibilitatea de a afișa **toate** intersecțiile posibile, nu doar prima găsită (dacă se dorește).

8. Bibliografie

1. *Russell, S., & Norvig, P.*, **Artificial Intelligence: A Modern Approach**, Prentice Hall, 2010 — capitole privind Constraint Satisfaction Problems.
2. *Gent, I., MacIntyre, E., Prosser, P., Walsh, T.*, **The Constrainedness of Search**, AAAI, 1996.
3. Documentația oficială .NET / C# pentru Windows Forms: https://learn.microsoft.com/en-us/dotnet
4. Documentația **Newtonsoft.Json**: https://www.newtonsoft.com/json (pentru încărcarea/serializarea datelor).

9.Contribuții

Moloman Laurențiu-Ionuț:

1. Cod backend C# (logică interfață în concordanță cu algoritmul, parsare JSON, etc) + Documentație

Petrișor Rareș-Gabriel:

2. Interfata + implementare/gandire Algoritm (ForwardChecking si ReduceDomains)

Proiect realizat în limbajul C#, framework .NET, cu interfață Windows Forms.