Coloriage de Graphe pour la Planification d'Événements

Slama Amr

9 décembre 2025

Résumé

Ce rapport présente une application de gestion d'événements qui utilise le problème de coloriage de graphe pour résoudre les conflits d'horaire. L'application permet aux utilisateurs de créer, visualiser et supprimer des événements, puis d'attribuer automatiquement des ressources (classes) aux événements qui se chevauchent en utilisant un algorithme de coloriage de graphe.

1 Introduction

La planification d'événements est un problème classique en informatique qui consiste à attribuer des ressources limitées (salles, professeurs, etc.) à des événements qui peuvent se chevaucher dans le temps. Ce problème peut être modélisé comme un problème de coloriage de graphe, où chaque événement est représenté par un sommet et les arêtes représentent les conflits (chevauchements) entre les événements.

L'application que nous avons développée offre une interface intuitive pour gérer les événements et résoudre automatiquement les conflits en utilisant un algorithme de coloriage de graphe.

2 Explication de l'Interface Utilisateur

L'application est construite avec PyQt6 et présente une interface moderne et intuitive composée de plusieurs éléments clés :

2.1 Menu de navigation latéral

Le menu de gauche permet aux utilisateurs de naviguer facilement entre différentes dates :

- Bouton `Open Calendar` pour ouvrir un calendrier et sélectionner une date spécifique
- Raccourcis pour naviguer rapidement vers `Today`, `Tomorrow`, `Overmorrow`, `Yesterday` et `Two Days Ago`

2.2 Zone principale

La zone principale affiche :
— La date actuellement sélectionnée
— Un tableau des événements pour cette date, avec des colonnes pour le nom de l'événement, sa durée et sa classe assignée
— Des boutons pour rafraîchir les événements, ajouter de nouveaux événements, supprimer l'événement sélectionné ou supprimer tous les événements

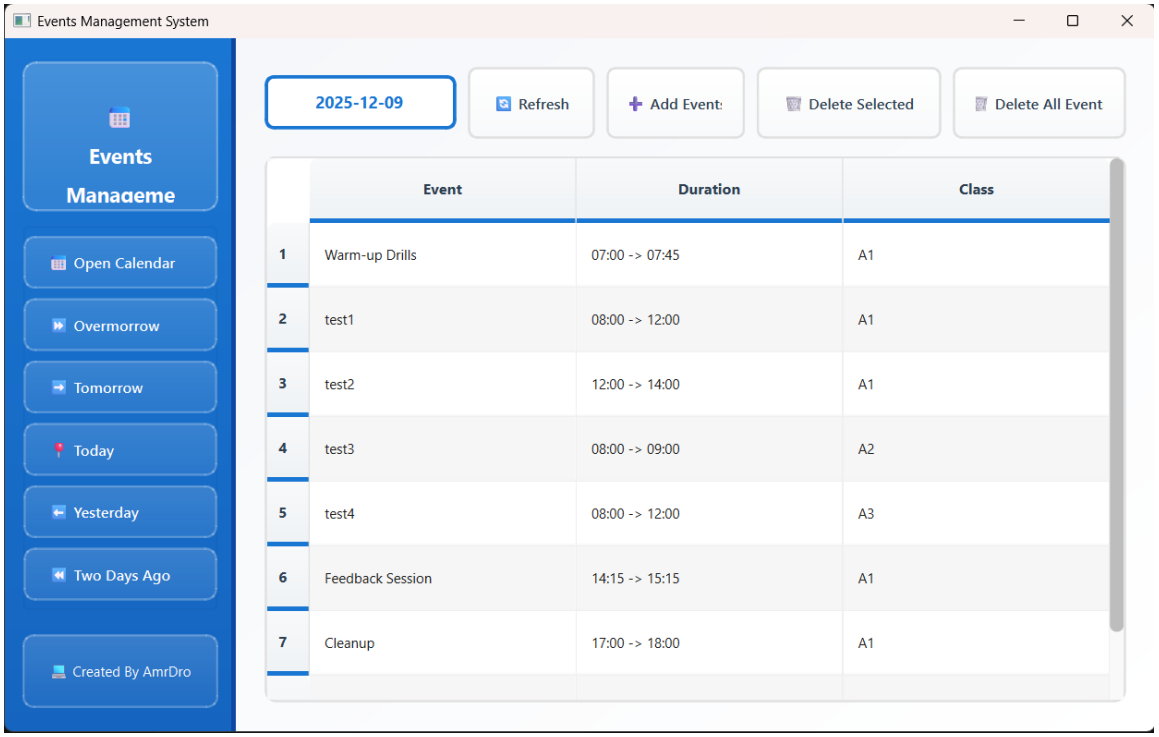

FIGURE 1 – Interface principale – tableau des événements et boutons d'actions

2.3 Boîte de dialogue d'ajout d'événement

Lorsqu'un utilisateur clique sur le bouton `Add Events`, une boîte de dialogue s'ouvre pour saisir :
— Le nom de l'événement
— L'heure de début et de fin

2.4 Calendrier interactif

Le bouton `Open Calendar` ouvre un calendrier graphique permettant de sélectionner rapidement n'importe quelle date.

2.5 Résolution des conflits

Le bouton `Refresh` déclenche l'algorithme de coloriage de graphe qui résout les conflits en assignant des classes différentes (A1, A2, etc.) aux événements qui se chevauchent.

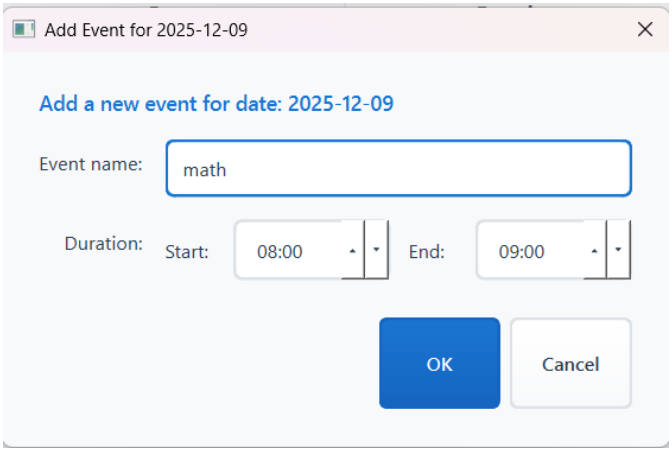

FIGURE 2 – Boîte de dialogue d'ajout d'un nouvel événement

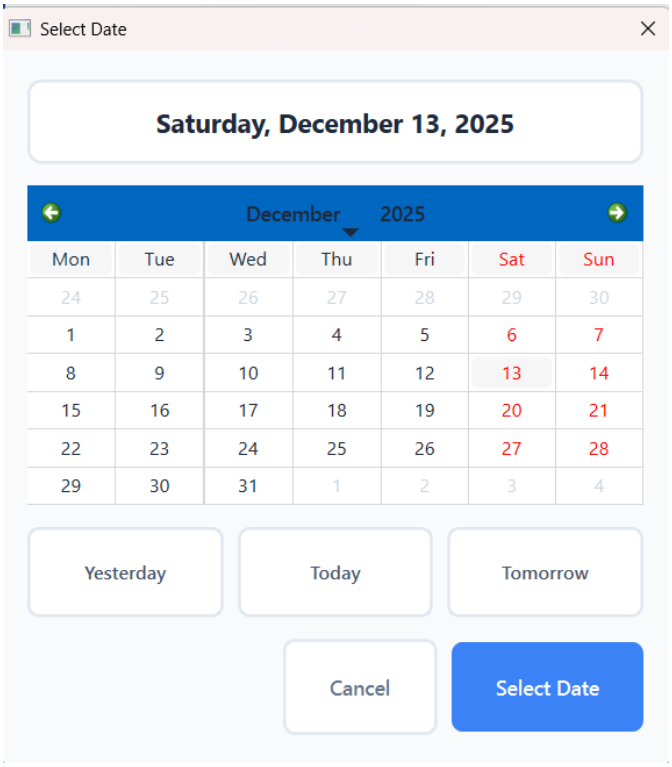

FIGURE 3 – Calendrier interactif pour la sélection rapide d'une date

3 Modélisation du Problème Mathématique

Le problème de planification d'événements peut être modélisé comme un problème de coloriage de graphe, qui est un problème NP-difficile en théorie des graphes.

3.1 Définition du problème

Étant donné un ensemble d'événements $E = \{e_1, e_2, ..., e_n\}$, chacun avec une période de temps $[s_i, f_i]$ (où s_i est l'heure de début et f_i l'heure de fin), nous voulons attribuer une classe (couleur) à chaque événement de telle sorte que :

— Si deux événements e_i et e_j se chevauchent dans le temps ($[s_i, f_i] \cap [s_j, f_j] \neq \emptyset$),

alors ils doivent avoir des classes différentes.
— L'objectif est de minimiser le nombre total de classes utilisées.

3.2 Construction du graphe

Nous construisons un graphe $G = (V, E)$ où :
— Chaque sommet $v \in V$ représente un événement.
— Une arête $(v_i, v_j) \in E$ existe si et seulement si les événements correspondants se chevauchent dans le temps.

3.3 Fonction de chevauchement

Deux événements e_i et e_j se chevauchent si :

$$\text{overlap}(e_i, e_j) = (s_i < f_j) \wedge (s_j < f_i)$$

3.4 Problème de coloriage

Le problème de coloriage de graphe consiste à trouver une fonction $c : V \to \{1, 2, ..., k\}$ telle que :

$$\forall (v_i, v_j) \in E, \quad c(v_i) \neq c(v_j)$$

Où k est le nombre de couleurs (classes) utilisées, que nous cherchons à minimiser.

3.5 Algorithme de coloriage glouton

L'application implémente un algorithme de coloriage glouton qui parcourt les sommets dans un ordre donné et assigne à chaque sommet la plus petite couleur disponible qui n'est pas utilisée par ses voisins :

```
colors = {}   # event -> color index

for v in V:
    forbidden = set()
    for (u, w) in E:
        if u == v and w in colors:
            forbidden.add(colors[w])
        if w == v and u in colors:
            forbidden.add(colors[u])

    c = 0
    while c in forbidden:
        c += 1

    colors[v] = c
```

Listing 1 – Algorithme de coloriage glouton

4 Résultats de l'Analyse

L'application a été testée avec divers scénarios pour évaluer l'efficacité de l'algorithme de coloriage.

4.1 Exemple de scénario

Considérons les événements suivants pour une journée :
- Événement A : 08 :00 - 10 :00
- Événement B : 09 :00 - 11 :00
- Événement C : 10 :00 - 12 :00
- Événement D : 13 :00 - 15 :00

Dans ce cas :
- Les événements A et B se chevauchent (09 :00 - 10 :00)
- Les événements B et C se chevauchent (10 :00 - 11 :00)
- Les événements A et C ne se chevauchent pas (A se termine à 10 :00, C commence à 10 :00)
- L'événement D ne chevauche aucun autre événement

4.2 Résultat du coloriage

L'algorithme de coloriage glouton produit le résultat suivant :
- Événement A : Classe A1
- Événement B : Classe A2
- Événement C : Classe A1 (car il ne chevauche pas A)
- Événement D : Classe A1 (car il ne chevauche aucun autre événement)

Ce résultat utilise 2 classes (A1 et A2), ce qui est optimal pour ce graphe.

4.3 Analyse de complexité

La complexité temporelle de l'algorithme de coloriage glouton implémenté est $O(n^2)$, où n est le nombre d'événements. Cette complexité est acceptable pour des applications pratiques avec un nombre modéré d'événements.

4.4 Limitations et améliorations possibles

L'algorithme de coloriage glouton ne garantit pas toujours une solution optimale en termes de nombre minimal de couleurs. Des améliorations pourraient inclure :
- L'utilisation d'heuristiques plus avancées pour l'ordre de traitement des sommets
- L'implémentation d'algorithmes exacts pour de petites instances
- L'intégration de solveurs d'optimisation comme Gurobi pour des solutions garanties optimales

5 Conclusion

L'application développée offre une solution efficace pour la gestion d'événements et la résolution de conflits d'horaire en utilisant le problème de coloriage de graphe. L'interface intuitive permet aux utilisateurs de gérer facilement leurs événements, tandis que l'algorithme de coloriage glouton résout automatiquement les conflits en attribuant des ressources appropriées.

Bien que l'algorithme actuel ne garantisse pas toujours une solution optimale, il offre un bon compromis entre efficacité et simplicité d'implémentation. Des améliorations futures pourraient inclure l'intégration de solveurs d'optimisation plus avancés pour garantir des solutions optimales.

Extensions possibles

— Intégration de contraintes additionnelles (capacité des salles, préférences des intervenants)
— Optimisation multi-objectif (minimiser les changements de salle, maximiser l'utilisation des ressources)
— Planification sur plusieurs jours avec contraintes de continuité
— Interface web pour l'accès multi-utilisateurs

Contribution personnelle

Cette application a été entièrement développée par **Slama Amr**, démontrant la maîtrise des concepts suivants :

— Développement d'interfaces graphiques avec PyQt6
— Modélisation de problèmes d'optimisation combinatoire
— Implémentation d'algorithmes de graphes
— Intégration de solveurs d'optimisation (Gurobi)
— Gestion de données persistantes (JSON)