Name: Kian Kolahdoozan

Matr.-Nr.: 46824

Problemstellung:

Das Wedding Seating Problem befasst sich mit der Verteilung von Gästen auf eine begrenzte Menge an Sitzplätzen bei einer Hochzeit. Dabei sollen die Beziehungen der Gäste berücksichtigt werden, damit alle Freunde an einem Tisch sitzen und die Feinde an einem anderen Tisch.

Das Ziel ist es, dass alle Gäste mit ihrem Platz zufrieden sind und Konflikte zwischen Gästen verhindert werden (Armas, 2024). Es ist ein Optimierungsproblem, bei welchem eine heuristische Lösung in Betracht kommt.

Vorgehensweise:

Die Datei mit den Gästen, Freunde und Feinde wird gelesen (Abbas, 2024).

Eine Relationshipmatrix wird anhand der Beziehungen erstellt (Lewis, 2013, S.3).

Die Liste mit den Gästen wird anhand der Anzahl der Beziehungen absteigend sortiert.

Der Gast mit dem meisten Beziehungen wird einem Tisch zugeordnet, deshalb "Most constraining variable" (MCV). Die Idee kam anhand des Skripts (Schumann, 2024, S.110).

Dann wird der Gast mit den zweitmeisten Beziehungen zugeordnet, unter der Bedingung, dass es kein Feind ist (no enemy).

Wenn der Tisch voll mit Gästen ist, die keine Feinde sind, wird bei jedem Gast überprüft, ob all die Freunde auch an dem Tisch sitzen. Wenn ja, dann ist der Tisch mit den Gästen fertig belegt und der nächste Tisch wird mit Gästen belegt.

Ich habe mich für MCV entschieden, da ich dachte, dass die mit den meisten Beziehung die größten Probleme bereiten, weshalb sie als Erstes einen Platz bekommen. Dabei wird erstmal bei der Zuordnung nur darauf geachtet, dass keine Feinde zusammensitzen, da sie wie die Freunde auch eine hohe Anzahl an Beziehungen haben und deshalb weiter vorne in der Liste als neutrale Kandidaten.

Dadurch sollte sich die Wahrscheinlichkeit erhöhen, dass Freunde zusammengesetzt werden. Mit der Nachbedingung, dass falls doch nicht alle Freunde zusammensitzen und der Bedingung, dass kein Gast an einen Tisch zuweisen darf, wenn an einem anderen Tisch ein Freund sitzt, wird garantiert, dass Freunde immer zusammensitzen. Es ist ein rekursiver Ansatz, orientiert an dem Skript (Schumann, 2024, S.108).

Im Nachhinein würde ich erst MRV und dann MCV zusammen nutzen, da es effizienter ist, den Gast mit den wenigsten Möglichkeiten als Erstes zu setzen und dann die anderen, die mehr Auswahl haben. Eine Kombination der heuristischen Funktionen wäre sinnvoll, falls bei einer der heuristischen

Funktionen zwei gleiche Werte rauskommen und die Reihenfolge nicht klar ist. Da kann die zweite heuristische Funktion die genaue Reihenfolge besser ermitteln.

Mit nur MCV funktioniert es auch, aber im schlimmsten Fall probiert der Code alle Möglichkeiten aus, bis es (k)eine Lösung findet. Bei diesem rekursiven Ansatz ist der Code sehr ineffizient, je größer die Anzahl an Gästen beträgt (GeeksforGeeks, 2024).

Ich habe beim Testen des Codes gesehen, dass es bei zu vielen Freunden keine Zuordnung findet, obwohl eine Zuordnung möglich wäre. Die genaue Ursache für diesen Bug habe ich nicht gefunden. Stattdessen habe ich einen anderen Ansatz genommen, wo die Freunde als eine Gruppe gespeichert werden und die Gruppe zusammen auf einem Tisch platziert wird. Diesen Ansatz habe ich in der Datei "Programm2" gespeichert, für das Testen den einkommentierten Code in Program.cs kopieren und ausführen.

In dem Ansatz wird MCV nicht mehr in dem ursprünglichen Ansatz genutzt, da die Anzahl der Beziehungen durch die Große der Gruppe ersetzt wird, wo aber die (freundlichen) Beziehungen eine Rolle spielen. Ich habe die folgenden Aufgaben nur für den Code in "Program.cs" beantwortet, weil das der ursprüngliche Code ist. Da kann man einfacher die Fragen beantworten, da sonst eine Variable kein Gast, sondern eine Gruppe von Gästen entspricht.

Hinweis: Falls ein Paar in der txt-Datei als Freunde und als Feinde eingegeben wird, werden die nur als Feinde dargestellt. Bei der manuellen Eingabe wird die Beziehung übernommen, die als Letztes hinzugefügt wurde. Der Programmablaufplan wurde als Bild und als Drawio Datei in den Ordner eingefügt, wo Progam.cs vorliegt.

Verwenden eines CSPs:

1) Wie wird das CSP modelliert

2) Wie wird ein State modelliert

3) Was gibt das Programm im Erfolgs/Fehlerfall aus

4) Welche Heuristische Funktionen werden verwendet

5) Wie modellieren Sie die friends/enemy Beziehung?

1)

17 Gäste und vier Tische mit jeweils fünf Plätzen

Variablen: {T1, T2, T3, T4, T5}

Domain: {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

T1 = {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

T2 = {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

T3 = {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

T4 = {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

T5 = {x01, x02, x03, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14, x15, x16, x17}

Beispielhafte Einschränkungen (Constrains), die dann ein Einfluss auf die Zuweisung der Tische haben:

Friends = {x01=x02, x12=x13, x06=x11} -> Tisch x01 = Tisch x02, Tisch x12 = Tisch x13, Tisch x06 = Tisch x11

Enemies = {x04!=x05, x14!=x16, x07!=x15} -> Tisch x04 != Tisch x05, Tisch x14 != Tisch x16, Tisch x07 != Tisch x15

2)

States:

T1 = {x01}

T2 = {}

T3 = {}

T4 = {}

T5 = {}

T1 = {x01, x03}

T2 = {}

T3 = {}

T4 = {}

T5 = {}

T1 = {x01, x03, x04}

T2 = {}

T3 = {}

T4 = {}

T5 = {}

T1 = {x01, x03, x04}

T2 = {x08}

T3 = {}

T4 = {}

T5 = {}

T1 = {x01, x03, x04, x05}

T2 = {x08}

T3 = {}

T4 = {}

T5 = {}

T1 = {x01, x03, x04, x05, x07}

T2 = {x08}

T3 = {}

T4 = {}

T5 = {}

usw. (sonst die PrintTables() Funktion in AssignSeatingHelper auskommentieren)

Änderungen verlaufen rekursiv durch AssignSeatingHelper und wird durch das Backtracking zurückgesetzt, falls es zu Konflikten kommt.

3)

Erfolgsfall: Seating arrangement found: {Ergebnis}

Fehlerfall je nach Fehlermeldung, sonst im Allgemeinen: Unable to seat all guests according to the given relationships.

4)

MCV

Gäste werden nach der Anzahl ihrer Beziehungen (Freunde und Feinde) absteigend sortiert (GetMCVList). Gäste mit den meisten Beziehungen (Constrains) werden zuerst gesetzt.

LCV wird indirekt genutzt, da kontrolliert wird, ob es ein Tisch mit ohne Konflikte ist.

5)

Die friends und enemies Beziehung werden in einer Matrix festgehalten. Dabei wurde die Matrix in der Literatur (Lewis, 2013, S.3) als ein Vorbild genommen.

Hier beispielsweise für fünf Gäste:

```
int[,] relationshipMatrix = new int[guests.Count, guests.Count]
{
  { 0,  1,  0, -1,  0 },     //  0 = neutral
  { 1,  0,  0,  0,  1 },     //  1 = friends
  { 0,  0,  0, -1,  0 },     // -1 = enemies
  {-1,  0, -1,  0,  0 },
  { 0,  1,  0,  0,  0 }
};
```

Internet- und Literaturquellen:

Abbas, M. M. (2024): Lesen und Schreiben in eine Datei in C#, https://www.delftstack.com/de/howto/csharp/read-and-write-to-a-file-in-csharp/, letzter Aufruf am 20.12.2024.

Armas, L. F. P. (2024): Mathematics of Love: Optimizing a Dining-Room Seating Arrangement for Weddings with Python, https://towardsdatascience.com/mathematics-of-love-optimizing-a-dining-room-seating-arrangement-for-weddings-with-python-f9c57cc5c2ce, letzter Aufruf am 20.12.2024.

GeeksforGeeks, (2024): Explain the Concept of Backtracking Search and Its Role in Finding Solutions to CSPs, https://www.geeksforgeeks.org/explain-the-concept-of-backtracking-search-and-its-role-in-finding-solutions-to-csps/, letzter Aufruf am 20.12.2024.

Lewis, R. (2013): Constructing Wedding Seating Plans: A Tabu Subject, Cardiff Univerity, 2013.

Schumann, E. (2024): Chapter III: Constrainst Satisfaction Problems, o.Ort, 2024.