

*Paweł Gogoliński
Grzegorz Ojdana
Krystian Trocha
II2-Z4*

Sprawozdanie z laboratorium sztucznej inteligencji Informatyka, sem. VI

I. Opis zadania

Napisać program w metodyce CLP(FD) rozwiązujący **Problem przydziału załóg lotniczych** (ang. *A crew allocation problem*), który zdefiniowany jest następująco:

Pewna linia lotnicza musi przydzielić swoich 20 pracowników do 10 lotów. Do każdego lotu potrzeba oddelegować określoną liczbę pracowników spełniających pewne założenia. Dla spełnienia oczekiwań klientów międzynarodowych w załodze każdego z lotów powinni być uwzględnieni pracownicy posługujący się odpowiednio językami niemieckim, hiszpańskim i francuskim. Dla każdego lotu określona jest także pewna minimalna liczba stewardów i stewardes. Ponadto każdemu z pracowników po odbytych locie należy się przerwa i nie może wziąć udziału w co najmniej dwóch następnych lotach.

II. Założenia realizacyjne

1. Założenia dodatkowe

Wejściem dla algorytmu rozwiązującego problem są dwa zbiory: zbiór pracowników określonych pewnymi cechami oraz zbiór wymagań poszczególnych lotów. Każdy z pracowników określony jest przez imię, pełnioną funkcję (steward bądź stewardesa) oraz języki, którymi się posługuje. Wymagania lotu zawierają liczbę załogi konieczną do odbycia lotu, minimalną liczbę stewardów i stewardes w załodze oraz minimalne liczby pracowników załogi mówiących w językach odpowiednio niemieckim, hiszpańskim i francuskim.

Wynikiem działania algorytmu jest zbiór załóg, tj. przypisań pracowników do konkretnych lotów. Do każdego lotu określonego wymaganiami wejściowymi musi zostać przydzielony zespół, który te wymagania spełnia.

Program skonstruowano tak, aby możliwe było przydzielenie załóg dla dowolnych danych wejściowych:

- użytkownik może zdefiniować personel dostępny do przydziału (imiona, pełnione funkcje oraz używane przez pracowników języki),
- użytkownik może zdefiniować wymagania lotów.

Dla zwiększenia atrakcyjności prezentacji wyników oraz ułatwienia wprowadzania danych, przygotowano graficzny interfejs użytkownika pozwalający na:

- wprowadzanie własnych danych testowych,
- użycie predefiniowanych danych,
- edycję cech personelu czy wymagań lotów,
- kilka form prezentacji wyników: w postaci listy zestawionych załóg oraz w formie grafu łączącego punkty symbolizujące pracowników i loty (użytkownik może wybrać, które loty i których pracowników chce umieścić na wykresie).

2. Metody, strategie oraz algorytmy wykorzystywane do rozwiązania zadania.

Dokładny opis realizacji struktur danych użytych w przedstawionych poniżej algorytmach zawarto w punkcie 4. Znak # przed operatorem arytmetycznym oznacza ograniczenie.

2.1. Algorytm wyznaczający załogi lotów

Dane:

Requires - lista wymagań następujących po sobie lotów.

$Requires = [R_1, \dots, R_N]$, $R_i = [TeamCount, Stewards, Stewardesses, French, German, Spanish]$

Crew - lista opisów personelu.

$Crew = [C_1, \dots, C_M]$, $C_i = [IsSteward, IsStewardess, SpeakFrench, SpeakGerman, SpeakSpanish]$

Wynik:

Team - macierz przypisań personelu do lotów.

$Team = [[T_{1,1}, \dots, T_{1,M}], [T_{2,1}, \dots, T_{2,M}], \dots, [T_{N,1}, \dots, T_{N,M}]]$, $T_{i,j} \in \{0, 1\}$, gdzie T_i odpowiada $Requires_i$, a $\forall_i T_{i,j}$ dotyczy $Crew_j$.

Metoda:

$matchTeams(Teams, Requires, Crew)$:

1. $NumFlights \# = \text{length}(Requires)$
2. $CrewCount \# = \text{length}(Crew)$
3. $ReqCrewCount := [Requires_{1,1}, Requires_{2,1}, \dots, Requires_{NumFlights, 1}]$
4. $FlightsReqs := Requires - ReqCrewCount$

5. `teamsMatrix(Teams, NumFlights, CrewCount, ReqCrewCount)`
{ opis algorytmu teamsMatrix umieszczony jest jako następny }
6. `checkFlightsReqs(Teams, FlightsReqs, Crew)` *{ Sprawdza, czy przygotowane załogi lotów spełniają wymagania }*
7. `flatten(Teams, TeamsList)`
8. `labeling(TeamsList)`

2.2. Algorytm wyznaczający macierz przypisań pracowników do lotów

Algorytm ten wyznacza jedynie jedno z możliwych przypisań pracowników do lotów, jednak nie dokonuje sprawdzenia czy wyznaczone załogi spełniają wymagania lotów. Jest to dokonywane przez osobny algorytm.

Dane:

NumFlights - liczba lotów, dla których należy wyznaczyć załogi.

CrewCount - liczność całego personelu linii lotniczej.

ReqCrewCount - lista zadanych liczności kolejnych załóg.

Wynik:

Teams - macierz przypisań członków personelu do załóg kolejnych lotów, taka sama, jak opisana w sekcji Wynik poprzedniego przedstawionego algorytmu.

Metoda:

`teamsMatrix(Teams, NumFlights, CrewCount, ReqCrewCount):`

1. Jeśli $NumFlights > 1$, wykonaj:
 - 1.1. $RestReqs := tail(ReqCrewCount)$
 - 1.2. `teamsMatrix(RestTeams, RestFlights, CrewCount, RestReqs)`
 - 1.3. $NumFlights \# = RestFlights + 1$
 - 1.4. Przyjmij listę $Team = [T_1, \dots, T_{CrewCount}]$, $T_i \in \{0, 1\}$
 - 1.5. $sum(Team) \# = ReqCrewCount$
 - 1.6. $Teams := [Team \mid RestTeams]$
 - 1.7. `checkBreaks(Teams)`
2. Jeśli $NumFlights = 1$:
 - 2.1. Przyjmij listę $Team = [T_1, \dots, T_{CrewCount}]$, $T_i \in \{0, 1\}$
 - 2.2. $sum(Team) \# = ReqCrewCount$
 - 2.3. $Teams = [Team]$

3. Języki programowania, narzędzia informatyczne i środowiska używane do implementacji systemu.

Implementację całego systemu można podzielić na następujące moduły:

- mechanizm rozwiązujący problem, napisany w języku Prolog (kompilacja SWI Prolog w wersji 6.6.2), z wykorzystaniem standardowej biblioteki *clpfd* (ang. *Constraint Logic Programming over*

Finite Domains). Jest to biblioteka umożliwiająca tzw. *programowanie z więzami* (tudzież z *ograniczeniami* - ang. *constraints*). Zakłada ono sformułowanie problemu jako zbioru ograniczeń, jakie jego rozwiązanie musi spełniać, oraz określenie dziedzin zmiennych. Takie podejście jest bardzo użyteczne przy tworzeniu rozwiązań problemów z dziedziny sztucznej inteligencji, problemów kombinatorycznych, przetwarzania języków naturalnych czy harmonogramowania,

- interfejs użytkownika, który przygotowano w języku *C#* z wykorzystaniem formatek *WinForms* dostępnych w środowisku *.Net 4.5.1*. Takie rozwiązanie pozwoliło łatwo i stosunkowo szybko przygotować przejrzysty interfejs użytkownika. Środowisko programistyczne, jakie wykorzystano do przygotowania interfejsu, to *Visual Studio 2012* w wersji *Professional*.

Narzędziem wspomagającym pracę nad projektem była usługa *Dysk Google* (ang. *Google Drive*, dostępna pod adresem <https://drive.google.com>) - jest to dysk sieciowy oraz zestaw narzędzi biurowych działających "w chmurze". Usługa ta wykorzystywana była do wspólnej pracy nad dokumentami w tym samym czasie, dzielenia się małymi plikami, a także do komunikacji.

III. Podział prac

Autor	Podzadanie
Paweł Gogoliński	Wdrożenie rozwiązania prologowego w aplikacji C#, opracowanie GUI.
Grzegorz Ojdana	Opracowanie rozwiązania problemu w Prologu.
Krystian Trocha	Przygotowanie interfejsu do połączenia rozwiązania w języku Prolog z aplikacją C#, opracowanie GUI.

IV. Opis implementacji

Część rozwiązania napisana w języku Prolog

1. Struktury danych

Jako dane wejściowe solver problemu napisany w Prologu przyjmuje listę opisów pracowników oraz listę wymagań lotów. Lista opisów pracowników złożona jest z krotek (list) o wartościach binarnych, definiujących kolejno: czy pracownik jest stewardem, czy jest stewardesą, czy mówi po francusku, czy mówi po niemiecku i czy mówi po hiszpańsku. Przykładowa definicja personelu:

```
Crew = [[1,0,0,0,1], % steward, mówi po hiszpańsku
        [1,0,1,1,0], % steward, zna francuski i niemiecki
        [0,1,0,0,0], % stewardesa, nie zna języków obcych]
```

```
[0,1,1,0,1]] % stewardesa, mówi po francusku i hiszpańsku
```

Lista wymagań lotów zawiera krotki o wartościach definiujących kolejno: licznosc personelu niezbędną do odbycia lotu, minimalną liczbę stewardów, minimalną liczbę stewardes, minimalną liczbę pracowników mówiących po francusku, niemiecku i hiszpańsku. Krotki zostały zaimplementowane jako listy. Przykład:

```
Requires = [[4, 1,1,1,1,1],  
            [5, 2,2,2,0,0],  
            [2, 1,0,0,1,1],  
            [6, 2,2,1,2,1]]
```

Wyjściem solvera jest macierz określająca przypisania personelu do lotów. Macierz ta została zaimplementowana jako lista list. Każdy wiersz stanowi tablicę o wartościach binarnych, gdzie wartość 1 w i -tym elemencie oznacza, że i -ty pracownik z listy wejściowej opisów personelu jest przypisany do załogi danego lotu. Przykład (nie jest powiązany z poprzednimi):

```
Teams = [[1, 0, 1, 0, 0, 0, 0], % lot 1: pracownicy nr. 1,3  
         [0, 1, 0, 0, 1, 0, 0], % lot 2: pracownicy nr. 2,5  
         [0, 0, 0, 1, 0, 1, 0], % lot 3: pracownicy nr. 4,6  
         [1, 0, 0, 0, 0, 0, 1]] % lot 4: pracownicy nr. 1,7
```

2. Predykaty

matchTeams(Teams, Requires, Crew)

Wejście:

Requires - wymagania lotów, opisane w sekcji *Struktury danych*.

Crew - opis personelu, również opisany we wspomnianej sekcji.

Wyjście:

Teams - macierz definicji załóg lotów w postaci opisanej w sekcji *Struktury danych*.

Działanie:

Predykat ten jest realizacją algorytmu *Wyznaczania załóg lotów* opisanego w rozdziale 2. Jest to predykat wywoływany w celu rozwiązania problemu będącego tematem zadania. Jego główną siłą jest użycie mechanizmu *labelingu*, który stanowi istotę korzystania z biblioteki *clpfd*, ponieważ pozwala na znalezienie rozwiązania poprzez ukonkretyzowanie zmiennych przekazanych poprzez listę w parametrze predykatu `labeling/2`. Stanowi to uruchomienie działania wewnętrznych mechanizmów programowania z więzami i wydajne znalezienie rozwiązania rozpatrywanego problemu opisanego w tej formie.

splitRequires(Requires, ReqCrewCount, FlightsReqs)

Wejście:

Requires - wymagania lotów, opisane w sekcji *Struktury danych*.

Wyjście:

ReqCrewCount - lista wymaganych liczności załóg.

FlightsReqs - lista pozostałych wymagań co do załóg.

Działanie:

Predykat ten wyluskuje z listy wymagań lotów pierwsze elementy każdej krotki, którymi są wymagane liczności załóg. Jest to predykat pomocniczy, a przygotowane w ten sposób listy wyjściowe są potrzebne do dalszych działań.

teamsMatrix(Teams, NumFlights, CrewCount, ReqCrewCount)

Wejście:

NumFlights - liczba lotów, którym należy przypisać załogi.

CrewCount - liczba wszystkich pracowników (liczność personelu).

ReqCrewCount - lista wymaganych liczności kolejnych załóg.

Wyjście:

Teams - macierz definicji załóg lotów w postaci opisanej w sekcji *Struktury danych*.

Działanie:

Predykat ten jest implementacją algorytmu *Wyznaczania macierzy przypisań pracowników do lotów*, opisanego w rozdziale 2. Wyznacza on załogi lotów i dba o to, by każda z nich miała określoną liczność. Ponadto zapewnione jest, że każdy z pracowników po odbytym locie nie jest przypisany do żadnej z załóg spośród dwóch następnych lotów.

checkFlightsReqs(Teams, FlightsRequirements, Crew)

Wejście:

Teams - macierz definicji załóg lotów w postaci opisanej w sekcji *Struktury danych*.

FlightsRequirements - wymagania co do lotów, w podobnej formie jak opisane w sekcji *Struktury danych*, pomniejszone o pierwszą kolumnę (czyli wszystkie wymagania oprócz wymogu liczności załogi).

Crew - opis personelu, opisany w sekcji *Struktury danych*.

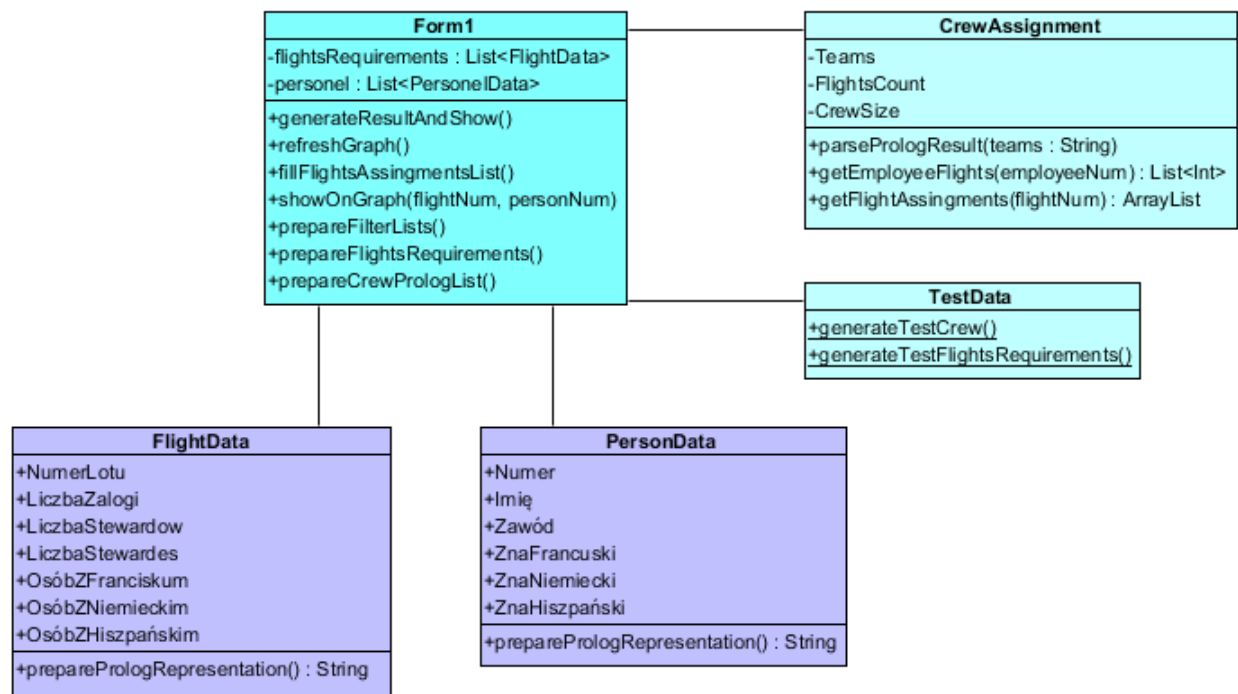
Wyjście:

Wartość *true*, jeśli podana macierz definicji załóg lotów spełnia wymagania przedstawione w parametrze *FlightsRequirements* dla personelu określonego przez parametr *Crew*. W przeciwnym wypadku, wartość *false*.

Działanie: Sprawdza, czy ustalone załogi lotów spełniają przedstawione przed nimi wymagania.

Część rozwiązania napisana w języku C#

Poniższy diagram klas przedstawia strukturę programu interfejsu użytkownika oraz struktury danych użyte do implementacji przepływu danych pomiędzy użytkownikiem a prologowym modułem solvera:



1. Struktury danych

Dane wejściowe dotyczące wymagań lotów oraz cech pracowników, pobrane z formatek interfejsu, trzymane są w postaci list przechowujących obiekty typu odpowiednio *FlightData* i *PersonData*. Dane te są trzymane w głównej klasie programu *Form1*.

2. Metody

`bool Form1.generateResultAndShow()`

Wejście: dane wymagań lotów i lista osób personelu, pobrane z list trzymanyh w klasie *Form1*.

Wyjście: wyświetlenie wyznaczonych załóg w oknie programu w formie listy lotów oraz w postaci grafu lub wyświetlenie informacji, że nie udało się znaleźć rozwiązania.

Działanie:

Metoda ta wywoływana jest, gdy użytkownik kliknie przycisk odpowiedzialny za zlecenie wygenerowania rozwiązania. Wewnątrz niej generowane jest rozwiązanie z wykorzystaniem modułu prologowego oraz jeśli udało się uzyskać rozwiązanie, przetwarzane jest ono przez obiekt klasy *CrewAssingment*. Następnie dane uzyskane w wyniku przetwarzania pobierane są z tego obiektu i przenoszone do formatek interfejsu.

`void CrewAssingment.parsePrologResult(String teams)`

Wejście: macierz przypisań personelu do lotów zapisana w formie ciągu znaków. Jest to rezultat zwrócony przez moduł prologowego programu rozwiązującego problem przydziału załóg.

Wyjście: zapis odczytanych z ciągu wejściowego danych do wewnętrznej macierzy liczb bądź zgłoszenie wyjątku, jeśli nie uda się poprawnie zinterpretować danych wejściowych.

Działanie:

Moduł prologowy programu zwraca macierz przypisań w formie łańcucha znaków, zatem wymagana jest zmiana reprezentacji uzyskanych danych na taką, którą można przedstawić użytkownikowi w atrakcyjnej formie. Metoda ta przegląda wejściowy ciąg znaków i próbuje odczytać kolejne wartości przypisań. Wartości binarne zapisywane są w wewnętrznej macierzy liczb *Teams*.

```
List<int> CrewAssingment.getEmployeeFlights(int employeeNum)
```

Wejście: numer pracownika.

Wyjście: lista numerów lotów, w których bierze udział pracownik.

Działanie:

Metoda ta wybiera te loty, do których przypisany jest pracownik określony numerem podanym na wejściu. Wywoływana jest podczas wyświetlania wyników przydziału załóg.

```
ArrayList CrewAssingment.getFlightAssignments(int flightNum)
```

Wejście: numer lotu.

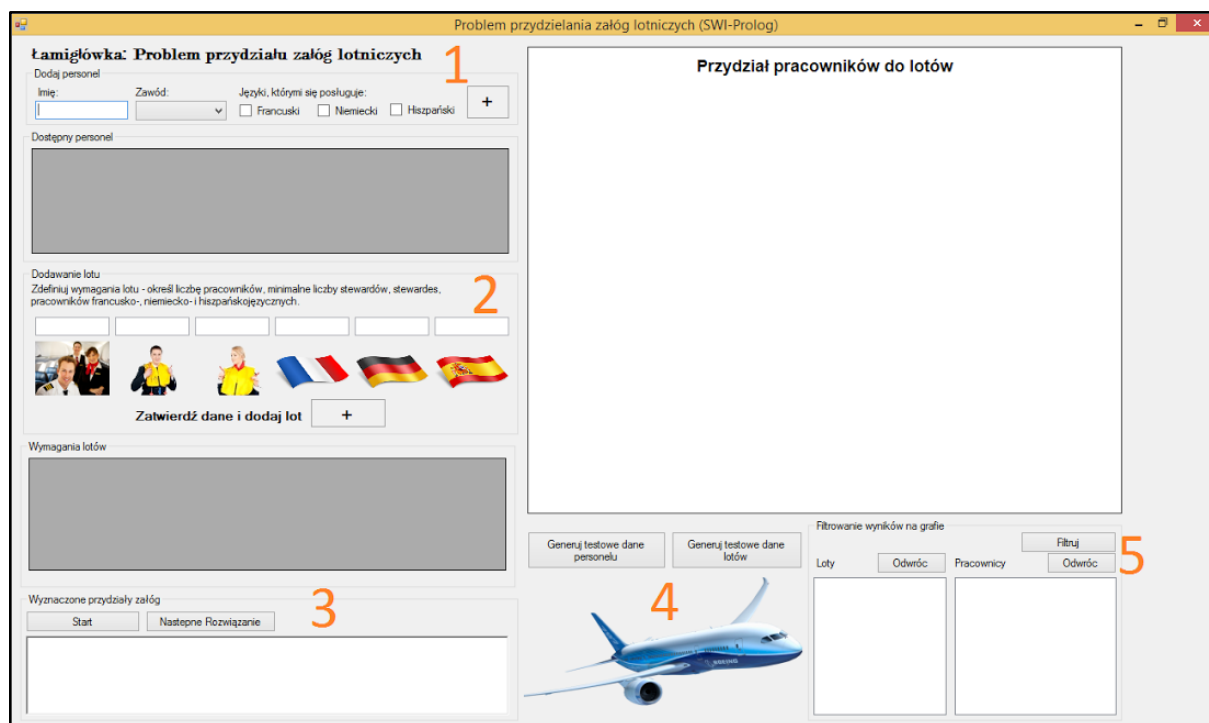
Wyjście: lista przypisań pracowników do lotu, zawierająca wartości 0 lub 1, gdzie 1 w *i*-tym elemencie listy oznacza, że do lotu przypisany jest pracownik o numerze *i*+1.

Działanie:

Metoda zwraca odpowiedni wiersz wewnętrznej macierzy *Teams*. Używana jest do pobrania danych po udanej interpretacji wyniku uzyskanego od modułu prologowego.

V. Użytkowanie i testowanie systemu

Okno programu przedstawiono na poniższym zrzucie ekranu. Numerami zaznaczono sekcje interfejsu (opisane poniżej). Ich kolejność odpowiada kolejności w jakiej użytkownik może ich użyć przy pracy z programem.



Sekcja 1: W tej sekcji można zdefiniować personalia pracownika: imię, zawód (do wyboru: steward lub stewardesa) oraz zaznaczyć języki, którymi się posługuje. Dodanie pracownika do personelu dokonywane jest po naciśnięciu przycisku ze znakiem plus (“+”, na rysunku oznaczony cyfrą 1). Wartości na liście pracowników można edytować, zatem łatwo jest poprawiać ewentualne pomyłki.

Sekcja 2: Definicji lotu dokonuje się poprzez wpisanie w pola tekstowe wymagań lotu. Kolejne pola od lewej strony oznaczają: dokładną licznosc załogi, minimalną liczbę stewardów w załodze, minimalną liczbę stewardes, minimalne liczby pracowników posługujących się odpowiednio językiem: francuskim, niemieckim, hiszpańskim. Podobnie jak przy dodawaniu pracownika, zatwierdzenie dodania lotu dokonywane jest przyciskiem “+”.

Sekcja 3: Gdy wprowadziliśmy już odpowiednie dane wejściowe (załoga oraz loty), możemy nakazać programowi przydzielić załogi do lotów za pomocą przycisku “Start”. W polu tekstowym pod przyciskiem “Start” pojawi się lista załóg. Klikając na przycisk “Następne Rozwiązanie”, solver podejmie próbę wygenerowania kolejnego rozwiązania. Po każdym wygenerowaniu rozwiązania rysowany jest graf przydziału (opisany poniżej w sekcji 5.).

Sekcja 4: W tej części okna programu znajdują się dwa przyciski służące automatycznemu wprowadzeniu predefiniowanych lotów i personelu. Ich naciśnięcie skutkuje wstawieniem stałego zestawu danych testowych do danych programu.

Sekcja 5: Sekcja ta pozwala filtrować przydziały pokazywane na grafie. Dokonuje się tego poprzez zaznaczenie bądź odznaczenie pól wyboru obok wyszczególnionych na liście numerów lotów oraz imion pracowników. Pole zaznaczone oznacza, że na grafie mają być narysowane połączenia dotyczące wybranego lotu bądź pracownika. Wciśnięcie przycisku “Filtruj” skutkuje odświeżeniem grafu, natomiast przycisk “Odwróć” powoduje odwrócenie stanu zaznaczenia wszystkich pól wyboru z danej listy.

Poniżej przedstawiono na przykładach sposób korzystania z programu. W pierwszej kolejności definiowany jest personel oraz wymagania lotów:

Problem przydział

Łamigłówka: Problem przydziału załóg lotniczych

Dodaj personel



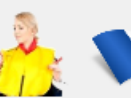



Imię: Zawód: Języki, którymi się posługuje: ☐ Francuski ☐ Niemiecki ☐ Hiszpański

Dostępny personel

	Numer	Imię	Zawód	ZnaFrancuski	ZnaNiemiecki	ZnaHiszpański
	1	Marian	Steward	T	N	T
	2	Tomasz	Steward	N	T	N
	3	Maria	Stewardesa	T	T	N
	4	Anna	Stewardesa	N	N	T

Dodawanie lotu

Zdefiniuj wymagania lotu - określ liczbę pracowników, minimalne liczby stewardów, stewardes, pracowników francusko-, niemiecko- i hiszpańskojęzycznych.

Zatwierdź dane i dodaj lot

Wymagania lotów

	Nume	LiczbaZalogi	LiczbaStewarda	LiczbaStewardes	OsóbZFrancu	OsóbZNiem	OsóbZHiszp
1	3	1	1	1	1	1	
2	2	1	0	1	0	1	
3	2	1	1	0	1	0	

Wymagania lotów wprowadzonych jako dane do programu widać na powyższym zrzucie ekranu. Natomiast pełną listę opisów pracowników przedstawia poniższa tabela:

Numer	Imię	Zawód	Zna francuski	Zna niemiecki	Zna hiszpański
1	Marian	Steward	T	N	T
2	Tomasz	Steward	N	T	N
3	Maria	Stewardesa	T	T	N
4	Anna	Stewardesa	N	N	T
5	Józef	Steward	N	T	T
6	Klara	Stewardesa	N	T	N
7	Genowefa	Stewardesa	T	T	N

Po naciśnięciu przycisku “Start” generujemy pierwsze możliwe rozwiązanie problemu. Przypisanie personelu do lotów przedstawione jest od razu w postaci listy załóg. Pierwsza linia poniższej listy oznacza: w załodze lotu nr. 1 znajdują się Józef, Klara i Genowefa.

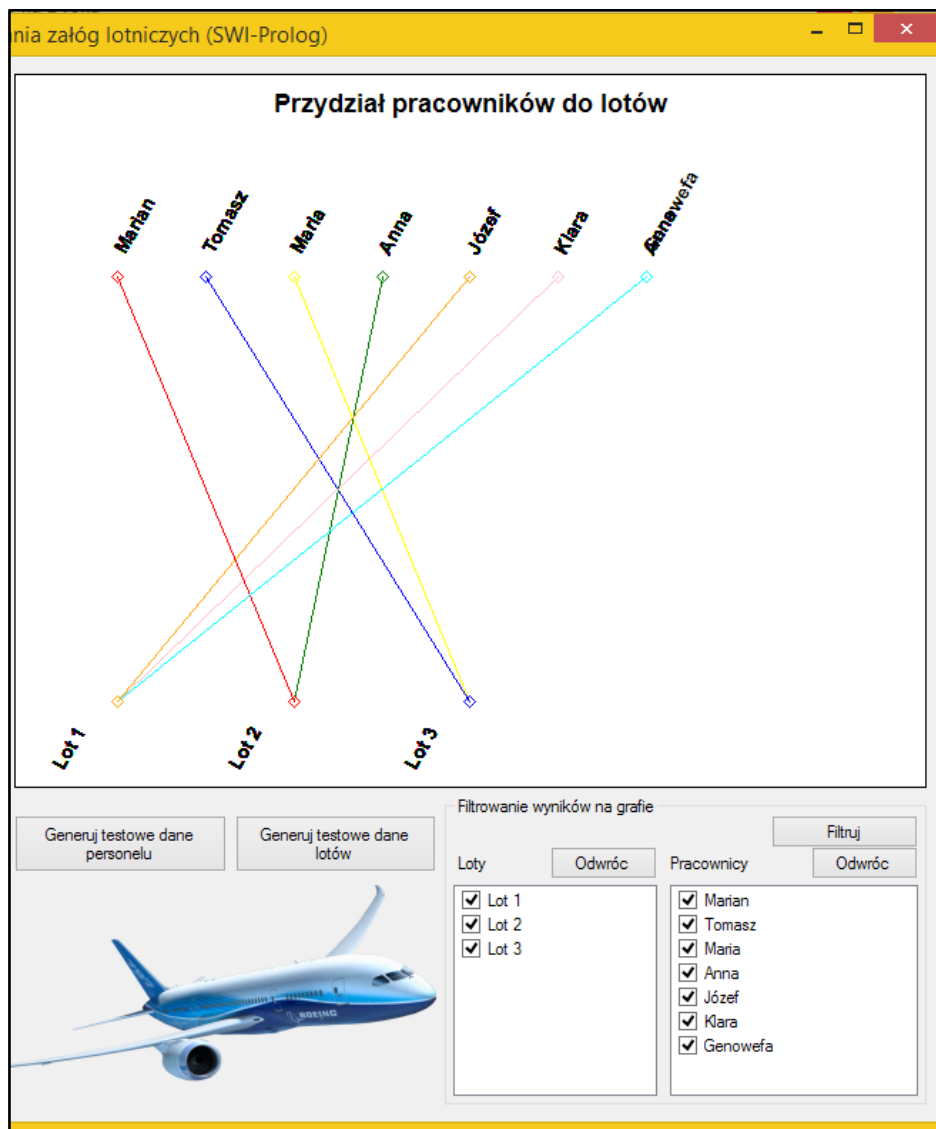
Wyznaczone przydziały załóg

Start

Następne Rozwiązanie

Lot 1: Józef Klara Genowefa
Lot 2: Marian Anna
Lot 3: Tomasz Maria

Graf przedstawiający przypisanie pracowników do lotów dla rozpatrywanego przykładu prezentuje się następująco:



W górnej części grafu znajdują się punkty oznaczające pracowników (podpisane są ich imionami). Punkty w dolnej części grafu symbolizują kolejne loty. Połączenie między lotem a pracownikiem wskazuje, że dany pracownik bierze udział w danym locie.

Jeśli uzyskane wyniki są dla użytkownika niesatysfakcjonujące, możliwe jest wygenerowanie innego rozwiązania poprzez kliknięcie przycisku “Następne Rozwiązanie”. Po jego naciśnięciu uzyskany zostanie następujący rezultat:

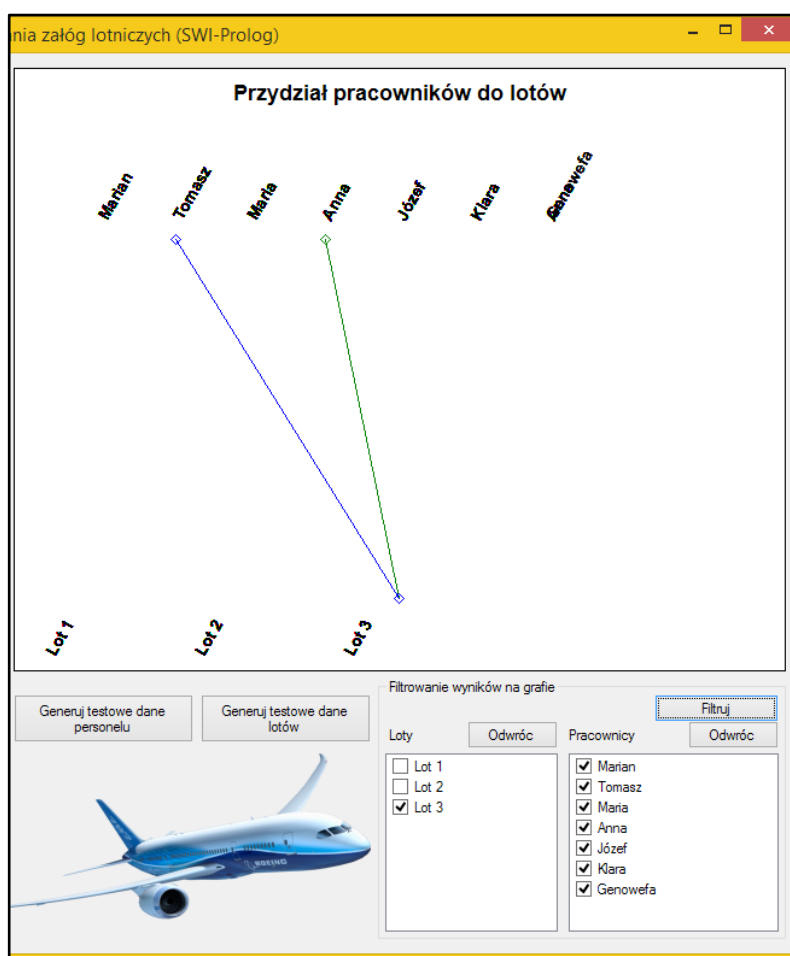
Wyznaczone przydziały załóg

Start **Następne Rozwiązanie**

Lot 1: Józef Klara Genowefa
 Lot 2: Marian Maria
 Lot 3: Tomasz Anna

Porównując uzyskany wynik z poprzednim, widać, że różnica jest niewielka - Maria, która przypisana była do lotu trzeciego, zamieniła się miejscami z Anną. Każde następne wyznaczanie załogi dla tych samych danych wejściowych powoduje właśnie takie niewielkie zmiany, co wynika bezpośrednio z mechanizmu działania prologowego solvera.

W przypadku, gdyby użytkownika interesowała tylko załoga lotu trzeciego, możliwe jest dostosowanie grafu tak, by znalazły się na nim przypisania pracowników tylko do tego lotu. Aby to uzyskać, w sekcji filtrów zaznaczyć należy tylko i wyłącznie pole wyboru obok etykiety "Lot 3". Po kliknięciu przycisku "Filtruj" odświeżany jest graf wynikowy. Sytuacja będzie wyglądać następująco:



Poniżej rozpatrzono inny przykład. Dane wejściowe przedstawiono na poniższym zrzucie ekranu:

Problem przydziel

Łamigłówka: Problem przydziału załóg lotniczych







Dodaj personel

Imię: Zawód: Języki, którymi się posługuje: ☐ Francuski ☐ Niemiecki ☐ Hiszpański

Dostępny personel

	Numer	Imię	Zawód	ZnaFrancuski	ZnaNiemiecki	ZnaHiszpański
	1	Grzegorz	Steward	N	T	N
▶	2	Paweł	Steward	T	T	N
	3	Krzysztof	Steward	N	N	T
	4	Monika	Stewardesa	T	N	N

Dodawanie lotu
Zdefiniuj wymagania lotu - określ liczbę pracowników, minimalne liczby stewardów, stewardes, pracowników francusko-, niemiecko- i hiszpańskojęzycznych.

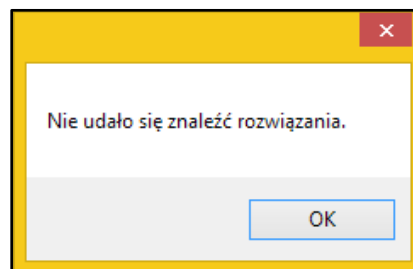







Zatwierdź dane i dodaj lot

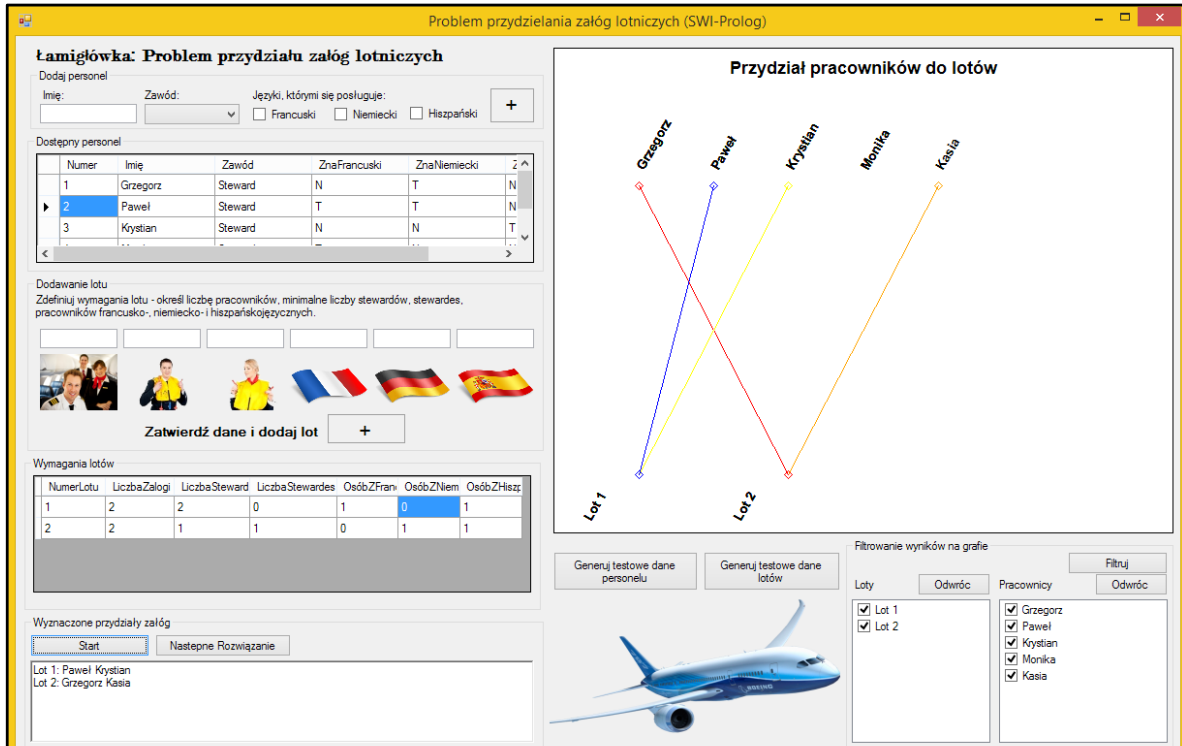
Wymagania lotów

	NumerLotu	LiczbaZalogi	LiczbaSteward	LiczbaStewardes	OsóbZFranc	OsóbZNiem	OsóbZHiszp
	1	2	2	0	1	0	1
	2	2	1	1	0	1	1

Próba wygenerowania rozwiązania dla takich danych skutkuje wyświetleniem następującego komunikatu:



Jest to spowodowane faktem, że dla podanych danych wejściowych niemożliwe jest znalezienie poprawnego rozwiązania. Aby to zmienić, do zbioru pracowników dodano stewardesę Kasię znającą język hiszpański. Teraz próba wygenerowania rozwiązania kończy się powodzeniem. Wynik przedstawiono na poniższym zrzucie ekranu.



VI. Tekst programu

Poniżej przedstawiono treść programu rozwiązującego problem przydziału załóg lotniczych w Prologu. Nie przedstawiono kodu części programu interfejsu użytkownika (napisanej w C#), ponieważ nie ma ona wpływu na rozwiązanie problemu (w części tej dostarczono jedynie moduł komunikacyjny między częścią prologową a zaprogramowaną w języku C# oraz graficzny interfejs użytkownika).

```
:- module(crewProb, [test/0, matchTeams/3]).
:- use_module(library(clpfd)).

matchTeams(Teams, Requires, Crew) :-
    length(Requires, NumFlights),
    length(Crew, CrewCount),

    splitRequires(Requires, ReqCrewCount, FlightsReqs),
    teamsMatrix(Teams, NumFlights, CrewCount, ReqCrewCount),
    checkFlightsReqs(Teams, FlightsReqs, Crew),

    flattenMatrix(Teams, TeamsList),
    labeling([ff,bisect,down], TeamsList)
.

splitRequires([], [], []).
splitRequires([[RE|RL]|RestCrew], [RE|RestEmps], [RL|RestLangs]) :-
    splitRequires(RestCrew, RestEmps, RestLangs).

teamsMatrix([Team], 1, CrewCount, [ReqCrewCount]) :-
    length(Team, CrewCount),
```

```

        Team ins 0..1,
        sum(Team, #=, ReqCrewCount).
teamsMatrix([Team|RestTeams], Flights, CrewCount, [ReqCrewCount|RestReqs]) :-
    teamsMatrix(RestTeams, RestFlights, CrewCount, RestReqs),
    Flights #= RestFlights+1,
    length(Team, CrewCount),
    Team ins 0..1,
    sum(Team, #=, ReqCrewCount),
    checkBreaks([Team|RestTeams]).

checkFlightsReqs([], [], _).
checkFlightsReqs([Team|RestTeams], [FlightsReq|RestReqs], Crew) :-
    checkEmployees(Team, TeamAttrbts, Crew),
    checkRelation(TeamAttrbts, FlightsReq, #>=),
    checkFlightsReqs(RestTeams, RestReqs, Crew).

checkEmployees([], [0,0,0,0,0], _).
checkEmployees([0|RT], TeamAttrbts, [_|Crew]) :-
    checkEmployees(RT, TeamAttrbts, Crew).
checkEmployees([1|RT], TeamAttrbts, [EAttr|Crew]) :-
    checkEmployees(RT, RAttr, Crew),
    addVector(RAttr, EAttr, TeamAttrbts).

checkBreaks([A,B]) :-
    addVector(A,B,AB),
    AB ins 0..1.
checkBreaks([A,B,C|R]) :-
    addVector(A,B,AB),
    AB ins 0..1,
    addVector(A,C,AC),
    AC ins 0..1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

test :-
    Crew =
    [[1,0,0,0,1],    % Tom    = 1
     [1,0,0,0,0],    % David  = 2
     [1,0,0,0,1],    % Jeremy = 3
     [1,0,0,0,0],    % Ron   = 4
     [1,0,0,1,0],    % Joe   = 5
     [1,0,1,1,0],    % Bill  = 6
     [1,0,0,1,0],    % Fred  = 7
     [1,0,0,0,0],    % Bob   = 8
     [1,0,0,1,1],    % Mario  = 9
     [1,0,0,0,0],    % Ed    = 10

     [0,1,0,0,0],    % Carol  = 11
     [0,1,0,0,0],    % Janet  = 12
     [0,1,0,0,0],    % Tracy  = 13
     [0,1,0,1,1],    % Marilyn = 14
     [0,1,0,0,0],    % Carolyn = 15
     [0,1,0,0,0],    % Cathy  = 16
     [0,1,1,1,1],    % Inez   = 17
     [0,1,1,0,0],    % Jean   = 18
     [0,1,0,1,1],    % Heather = 19
     [0,1,1,0,0]     % Juliet = 20
    ],

    CrewNames = [tom, david, jeremy, ron, joe,
                  bill, fred, bob, mario, ed,
                  carol, janet, tracy, marylin, carolyn,
                  cathy, inez, jean, heather, juliet],

    Requires = [[4, 1,1,1,1,1],

```



```

[5, 1,1,1,1,1],
[5, 1,1,1,1,1],
[6, 2,2,1,1,1],
[7, 3,3,1,1,1],
[4, 1,1,1,1,1],
[5, 1,1,1,1,1],
[6, 1,1,1,1,1],
[6, 2,2,1,1,1],
[7, 3,3,1,1,1]
],

matchTeams(Teams, Requires, Crew),

nl,
printTeams(Teams, CrewNames),
nl
.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

checkRelation([], [], _).
checkRelation([A|AR], [B|BR], Rel) :-
    call(Rel,A,B),
    checkRelation(AR, BR, Rel).

addVector([], [], []).
addVector([A|RA], [B|RB], [C|Res]) :-
    addVector(RA,RB,Res),
    C #= A+B.

flattenMatrix([T], T).
flattenMatrix([T|RT], TL) :-
    flattenMatrix(RT, RTL),
    append(T, RTL, TL).

printTeams(Teams, CrewNames) :-
    printTeams(Teams, 1, CrewNames).
printTeams([T], Nr, Names) :-
    format('Lot ~a: ', Nr),
    printTeam(T, Names).
printTeams([T|R], Nr, Names) :-
    format('Lot ~a: ', Nr),
    printTeam(T, Names),
    Nr2 is Nr+1,
    printTeams(R, Nr2, Names).

printTeam([], _) :- nl.
printTeam([O|R], [_|RNames]) :-
    printTeam(R, RNames).
printTeam([1|R], [Name|RNames]) :-
    format('~w ', Name),
    printTeam(R, RNames).

```

Bibliografia

- [1] Opis i wyjaśnienie zadanego problemu dostępny na stronie <http://mozart.github.io/mozart-v1/doc-1.4.0/fst/node5.html#fset.examples.crew> (ostatni dostęp: 14 czerwca 2014).
- [2] Oficjalna dokumentacja biblioteki *clpfd* dla środowiska SWI Prolog, dostępna pod adresem <http://www.swi-prolog.org/man/clpfd.html> (ostatni dostęp: 14 czerwca 2014).
- [3] Joseph Albahari, Ben Albahari, *C# 4.0 in a nutshell*, wyd. 4, O'Reilly Media Inc., 2010
- [4] Serwis *msdn.microsoft.com* jako dokumentacja języka *C#*, platformy *.Net* oraz technologii *WinForms*: [http://msdn.microsoft.com/en-us/library/w0x726c2\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/w0x726c2(v=vs.110).aspx) (ostatni dostęp: 14 czerwca 2014).