dr inż. Paweł Sitek, dr inż. Jarosław Wikarek
Politechnika Świętokrzyska

ZASTOSOWANIE ŚRODOWISK DEKLARATYWNYCH DO WSPOMAGANIA DECYZJI HARMONOGRAMOWANIA PRODUKCJI

Problemy harmonogramowania pojawiają się na wielu poziomach problemów decyzyjnych dotyczących produkcji. Na ogół charakteryzują się dużą liczbą ograniczeń (kolejności wykonania zadań, dostępnych zasobów itp.). Ze względu na złożoność obliczeniową wynikającą z dużej liczby zmiennych decyzyjnych całkowitoliczbowych oraz charakteru problemów są klasyfikowane jako zadania NP-trudne. Z tego powodu tradycyjne podejścia do rozwiązania tych problemów opierające się na programowaniu matematycznym są często nieefektywne. W odróżnieniu od podejścia tradycyjnego gdzie modelowanie ograniczeń problemu zwykle jest sztuczne (użycie dodatkowych zmiennych 0-1) w środowisku programowania w logice z ograniczeniami (CLP – Constrain Logic Programming) modelowanie ograniczeń jest naturalne i wynika z paradygmatu na którym środowisko CLP zostało oparte. Środowisko CLP jest środowiskiem deklaratywnym. W pracy zaproponowano wykorzystanie środowisk deklaratywnych (CLP, SQL, HTML) do budowy systemu wspomagania decyzji harmonogramowania produkcji.

IMPLEMENTATION OF DECLARATIVE FRAMEWORK FOR DECISION SUPPORT IN SCHEDULING PROBLEMS

Scheduling problems appear frequently at different levels of decisions. They are usually characterized by many types of constraints, which make them unstructured and difficult to solve (NP-complete). Traditional mathematical programming approaches are deficient because their representation of constraints is artificial (using 0-1 variables). Unlike traditional approaches, constraint logic programming (CLP) provides for a natural representation of heterogeneous constraints. In CLP we state the problem requirements by constraints; we do not need to specify how to meet these requirements. In this paper we propose a declarative framework for decision support system (DSS) for scheduling problems implemented by CLP and relational SQL database. We illustrate this concept by the implementation of a DSS for scheduling problems with external resources in different production organization environments.

1. WPROWADZENIE

Problemy szeregowania zadań i rozdziału zasobów należą do najczęstszych zagadnień, których rozwiązanie jest niezbędne przy podejmowaniu decyzji w obszarze planowania i harmonogramowaniu produkcji. Mają także zastosowanie w sieciach komputerowych, systemach operacyjnych, realizacji projektów badawczych, konstrukcyjnych, ale również przy układaniu planów zajęć, dyżurów, remontów itp.
Z formalnego punktu widzenia powyższe problemy można scharakteryzować w następujący sposób. Dany jest zbiór zadań, zbiór maszyn (w ogólności procesorów) oraz zwykle ograniczone zasoby przeznaczone do wykonania tych zadań. Należy znaleźć takie dopuszczalne przyporządkowanie zadań do maszyn (procesorów) oraz taką kolejność wykonania zadań na maszynach oraz taki dopuszczalny rozdział ograniczonych zasobów pomiędzy zadania/maszyny, aby przyjęte kryterium efektywności osiągnęło optimum.

Ze względu na możliwe do uzyskania efekty ekonomiczne są podejmowane rozliczne projekty badawcze oraz próby implementacji systemów wspomagających decyzje szeregowania zadań i rozdziału zasobów. Kłopoty implementacyjne wynikają głównie z dwóch powodów. Po pierwsze powyższe problemy należą do bardzo trudnych zagadnień optymalizacji dyskretnej, które charakteryzują się trudnościami natury obliczeniowej (np. eksplozja kombinatoryczna, duża złożoność obliczeniowa- zadania NP-trudne). Drugim istotnym utrudnieniem jest duża liczba typów tych problemów, które różnią się sposobem organizacji, rodzajem maszyn (procesorów), ograniczeniami kolejnościowymi, występowaniem lub nie przezbrojeń itd. Klasyfikację i szczegółowy opis tych zagadnień można znaleźć m.in. w [1]. W zależności od złożoności obliczeniowej poszczególnych problemów konstruowane są optymalne algorytmy wielomianowe (niezwykle rzadko), aproksymacyjne algorytmy wielomianowe bądź algorytmy metaheurystryczne wykorzystujące elementy sztucznej inteligencji typu: sieci neuronowe, algorytmy genetyczne, tabu-search itd. Dopuszcza się również konstruowanie tzw. wykładniczych algorytmów optymalnych wykorzystujących metodę branch-and-bound oraz programowania dynamicznego [1].

Biorąc pod uwagę powyższe aspekty a w szczególności złożoność obliczeniową oraz dużą liczbę i zróżnicowanie problemów szeregowania zadań i rozdziału obciążeń, która skutkuje koniecznością opracowywania wielu algorytmów i metod rozwiązywania dla poszczególnych szczegółowych problemów, zaproponowano podejście oparte na środowiskach deklaratywnych. Środowiska te charakteryzują się prostotą opisu i modelowania problemu, nie wymagają szczegółowych algorytmów i sposobu ich rozwiązywania. Maja natomiast wbudowane mechanizmy poszukiwania rozwiązań niezależne od programisty. W oparciu o środowiska deklaratywne zaproponowano system wspomagania decyzji dla problemów harmonogramowania produkcji.

2. ŚRODOWISKA DEKLARATYWNE

Środowiska deklaratywne są oparte na paradygmacie programowania deklaratywnego. W środowiskach tych program składa się najczęściej nie z podstawień sterowania przepływem (pętle instrukcje warunkowe itp.) lecz z deklaracji. W odróżnieniu od języków imperatywnych gdzie należy znaleźć sposób rozwiązania problemu (algorytm) czyli odpowiedzieć na pytanie „jak rozwiązać problem” w językach deklaratywnych odpowiadamy na pytanie „jaki jest problem”.

Do języków deklaratywnych możemy zaliczyć zatem języki skryptowe opisujące wygląd stron takie jak np. HTML, XML, jak również język zapytań baz danych SQL (Structure Query Language). W języku SQL w podstawowej instrukcji *select* specyfikuje się jakie dane z jakich tabel i pod jakim warunkiem chcemy uzyskać a nie w jaki sposób. Drugą grupę języków deklaratywnych stanowią języki programowania w logice. W językach tych zapisuje się stwierdzenia używając rachunku predykatów pierwszego rzędu. Wyniki powstają jako rezultat (automatycznego) wnioskowania. Jednym masowo rozpowszechnionym językiem programowania w logice jest język PROLOG wykorzystujący resolucję jako podstawą metodę wnioskowania. Język PROLOG powstał w latach siedemdziesiątych ubiegłego wieku. Opracowano go z myślą o przetworzeniu języka naturalnego i matematycznym dowodzeniem twierdzeń. Inną cechą języków programowania w logice jest ich naturalna równoległość czyli brak podstawowego założenia języków imperatywnych jako sekwencyjnej maszyny Von Neumana. Programowanie w logice od tego czasu rozwinęło się. Jednym z kierunków było dodanie do paradygmatu programowania w logice paradygmatu programowania z ograniczeniami. W wyniku tej modyfikacji powstało środowisko programowania w logice

z ograniczeniami(CLP – Constraint Logic Programming) [2] gdzie prostą zasadę rezolucji zastąpiono problemem spełnienia ograniczeń CSP (Constrain Satisfaction Problem).

2.1. PROBLEM SPEŁNIENIA OGRANICZEŃ –CSP (CONSTRAINT SATISFACTION PROBLEM)

Podstawowym zagadnieniem w programowaniu w logice z ograniczeniami jest tzw. problem spełnienia ograniczeń – Constraint Satisfaction Problem – CSP [3]. Można ogólnie określić go w następujący sposób. Dany jest skończony zbiór zmiennych, każda zmienna posiada skończoną domenę wartości, jakie może przyjąć. Dany jest również zbiór ograniczeń, który nakłada określone warunki na wartości, jakie zmienne mogą przyjmować. Rozwiązaniem CSP jest znalezienie takiego przyporządkowania wartości do każdej zmiennej (ukonkretnienie zmiennej), aby wszystkie ograniczenia były jednocześnie spełnione.

Formalnie, dla danego problemu CSP z n zmiennymi decyzyjnymi X_1, …, X_n i m ograniczeniami R_1, …, R_m rozwiązanie to takie odwzorowanie $f(X_i) \rightarrow d_i$, $d_i \in D_i$, gdzie D_i jest domeną X_i, dla i=1..n, które dla każdego j, spełnia $R_j(f(X_{1i}), \ldots, f(X_{ki}))$. Można wyróżnić kilka kategorii problemów CSP są to [4]:

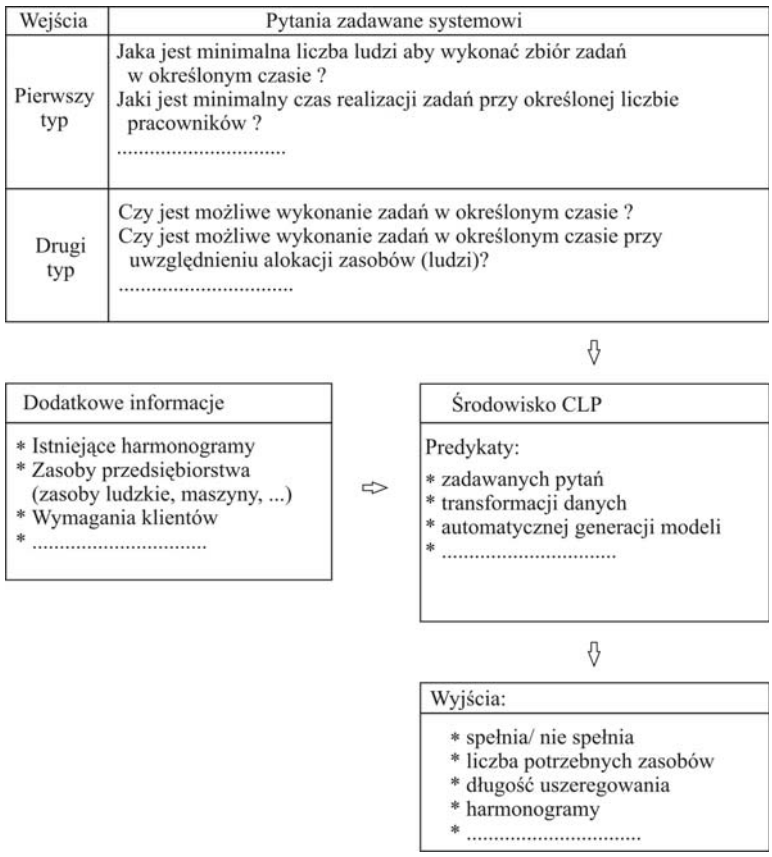
- CSP dla którego rozwiązaniem jest pojedynczy zbiór ukonkretnionych zmiennych spełniających wszystkie ograniczenia.
- CSP do którego rozwiązania należy znaleźć wszystkie zbiory ukonkretnionych zmiennych spełniających wszystkie ograniczenia.
- CSP do którego rozwiązania należy znaleźć zbiór ukonkretnionych zmiennych spełniających wszystkie ograniczenia i jednocześnie optymalizujący wskaźnik jakości.

W programowaniu z ograniczeniami każde ograniczenie R problemu CSP jest rozpatrywane jako podproblem. Z każdym ograniczeniem związany jest algorytm redukcji domen, który usuwa te wartości z domen, które są niedopuszczalne dla zmiennych decyzyjnych występujących w ograniczeniu. Dodatkowo w celu zwiększenia efektywności jest ułatwiona komunikacja pomiędzy ograniczeniami. Podstawową metodą używaną w tym celu jest propagacja ograniczeń (constraint propagation), która wiąże ograniczenia poprzez ich wspólne zmienne doprowadzając do wysokiego stopnia zgodności, poprzez usunięcie wartości niedopuszczalnych. Jest to osiągane poprzez systematyczną redukcję domen zmiennych decyzyjnych.

3. ZAŁOŻENIA SYSTEMU WSPOMAGANIA DECYZJI HARMONOGRAMOWANIA PRODUKCJI

Biorąc pod uwagę różnorodność problematyki harmonogramowania produkcji (organizacja produkcji job-shop, flow-shop, open-shop, project itp.) oraz złożoność obliczeniową modelowanych problemów zaproponowano wykorzystanie środowisk deklaratywnych do budowy systemu wspomagania decyzji dla problemów harmonogramowania produkcji. Ogólne założenia systemu pokazano na rys 1. Wejściem do systemu jest zbiór zadawanych pytań. Zadawane pytania mogą być dwojakiego typu. Pierwszy typ pytań dotyczy poszukiwania rozwiązania spełniającego określone kryterium jakości np. *Jaka jest minimalna liczba ludzi do wykonania zbioru zadań ?* Drugi typ pytań dotyczy istnienia rozwiązań np. *Czy jest możliwe uszeregowanie zbioru zadań w określonym czasie?* Sercem systemu jest środowisko CLP składające się z predykatów: implementujących zbiór pytań wejściowych, predykatów trans-

formacji danych z bazy danych oraz predykatów automatycznej generacji modeli CLP, które wykorzystują dwa pierwsze typy predykatów.



Rys. 1. Ogólna koncepcja systemu wspomagania decyzji harmonogramowania produkcji

Wyjściem systemu są odpowiedzi na zadane pytania (odpowiednio sformułowane predykaty) i odpowiadające im harmonogramy. W prezentowanej wersji systemu wspomagania decyzji harmonogramowania produkcji mogą, być zadawane między innymi następujące pytania (wywoływane odpowiednie predykaty CLP):

- Jaka jest minimalna liczba pracowników potrzebna do wykonania zadań w określonym czasie, i odpowiadający temu harmonogram? (predykat *opc_d(L,C)*).
- Jaki jest minimalny czas uszeregowania i odpowiadający mu harmonogram przy określonej liczbie pracowników? (predykat *opc_g(L,C)*).
- Czy jest możliwe wykonanie zbioru zadań w określonym terminie przy określonej liczbie pracowników? (predykat *opc_s(L,C))*.
- Jaki jest czas uszeregowania i odpowiadający mu harmonogram dla zbioru dodatkowych zadań? (predykat *opcd_g(L,C)*).
- Jaka jest potrzebna minimalna liczba pracowników dla uszeregowania zbioru dodatkowych zadań w zadanym czasie bez zmiany istniejącego harmonogramu (predykat *opcd_d(L,C)*).
- Czy jest możliwe uszeregowanie zadań w określonym czasie przy określonej liczbie pracowników? (predykat *opcd_s(L,C)*).

  $C=C_{max}$- długość uszeregowania , L-liczba pracowników

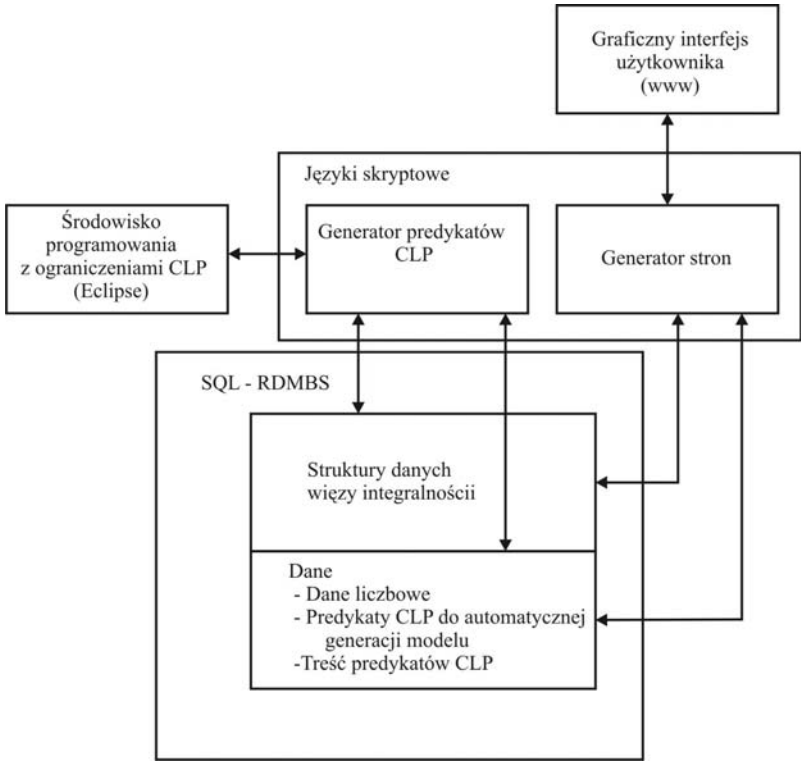
Powyższe pytania mogą uwzględniać dowolny zbiór dodatkowych zasobów (w tym przypadku są to tylko pracownicy) oraz dotyczyć dowolnej formy organizacji produkcji (job-shop, flow-shop, open-shop, project itp.). Dodatkowo w przedstawianej wersji systemu wspomagania decyzji została zaimplementowana dodatkowa funkcjonalność polegająca na rozdziale obciążeń. W problemach harmonogramowania opisanych w literaturze czas realizacji poszczególnych zadań jest na ogół stały i określony przed ich wykonaniem. W zastosowaniach praktycznych bardzo często czas wykonania poszczególnych zadań zależy w istotny sposób od ilości przydzielonych zasobów do ich wykonania. Zależności te na ogół są nieliniowe można je przedstawić za pomocą relacji (tabeli relacyjnej bazy danych). W systemie zaimplementowano możliwość zmiany czasu wykonania zadań w zależności od przydzielonej liczby pracowników. Powyższa funkcjonalność nie wymaga zmiany predykatów a jedynie odpowiednio przygotowanych danych opisujących problem, znajdujących się w relacyjnej bazie danych. Zaproponowana struktura relacyjnej bazy danych oraz sposób budowy predykatów CLP powoduje, że w systemie można generować nie tylko harmonogramy o określonych parametrach dla różnych form organizacji produkcji, ale również uwzględniać rozdział dodatkowych zasobów (w ogólnym przypadku zbiorów zasobowych) oraz różnego ich wpływu na wykonywane zadania.

4. IMPLEMENTACJA SYSTEMU WSPOMAGANIA DECYZJI HARMONOGRAMO-WANIA PRODUKCJI

Do implementacji systemu zastosowano środowisko deklaratywne. System wykonano w architekturze wielowarstwowej (Rys. 2). W warstwie bazy danych zastosowano relacyjną bazę danych z deklaratywnym językiem SQL – dostępu do danych. W warstwie logiki przetwarzania zastosowano środowisko CLP – Eclipse. Interfejs użytkownika oprogramowano w języku HTML. Strony są generowane dynamicznie. Zastosowanie środowisk deklaratywnych w każdej warstwie oraz modelu wielowarstwowego przynosi wiele zalet. Do najważniejszych z nich należy zaliczyć:

- Oddzielenie modelu od danych.
- Możliwość zdalnego korzystania z systemu (poprzez sieć WAN, Intranet, Internet).
- Łatwą modyfikację i rozbudowę części logiki systemu poprzez dodanie nowych predykatów.
- Możliwość zaprojektowania schematu bazy danych, w którym mogą być zapisane dane dla różnych typów problemów harmonogramowania produkcji.
- Łatwą modyfikację modelu CLP poprzez dodanie nowych ograniczeń (bez konieczności modyfikacji modeli już istniejących).
- Małe wymagania, co do sprzętu i oprogramowania, które występują po stronie klienta systemu– wystarczy komputer z przeglądarką internetową.

Istotną nowością prezentowanej implementacji jest integracja dwóch środowisk deklaratywnych RDBMS (Relational Database Management System) oraz CLP. Integracja ta polega między innymi na możliwości automatycznej generacji skryptów CLP na podstawie danych zawartych w bazie danych. Tak wygenerowane skrypty zawierają wiele predykatów CLP stanowiących kompletny model problemu harmonogramowania produkcji, który może być przesłany do silnika CLP.



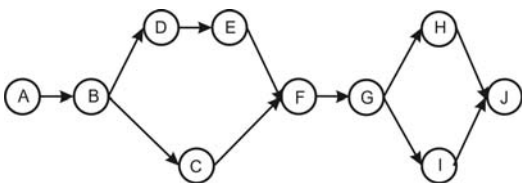
Rys. 2. Schemat implementacyjny systemu wspomagania decyzji

5. PRZYKŁADY ILUSTRACYJNE

W celu zilustrowania możliwości zaproponowanego systemu wspomagania decyzji harmonogramowania produkcji przedstawiono dwa przykłady liczbowe. Przykłady dotyczą problemów harmonogramowania dla różnych form organizacji produkcji. Pierwszy przykład dotyczy produkcji unikalnej, jednostkowej (project) np. budowy domu, statku, mostu itp. Drugi przykład jest typowy dla produkcji powtarzalnej odbywającej się w środowisku gniazdowym (job-shop). Dla zilustrowania rozdziału zasobów i ich wpływu na harmonogram wykonania przykład pierwszy został opracowany w dwóch wersjach. W pierwszej wersji (*przykład_1a*) czasy wykonania zadań są stałe w drugiej (*przykład_1b*) zależą od ilości przydzielonych zasobów dodatkowych (tu: pracowników). Dane liczbowe do *przykład_1a* przedstawiono w tab. 1. Na rys. 3 przedstawiono sieć kolejności wykonania zadań dla obu wersji przykładu pierwszego.

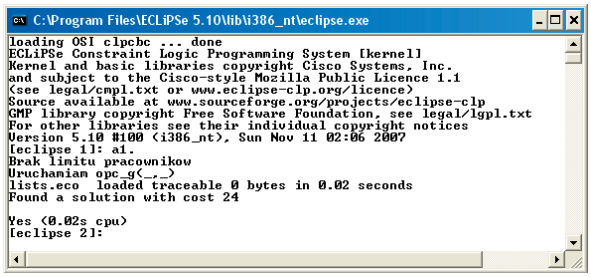
Tab. 1. Dane liczbowe *przykład_1a*

Operacja	Czas	Poprzednik	Pracownicy
A	2	-	4
B	3	A	8
C	4	B	4
D	3	B	3
E	2	D	3
F	3	C, E	6
G	4	F	8
H	5	G	4
I	3	G	6
J	2	H, I	8

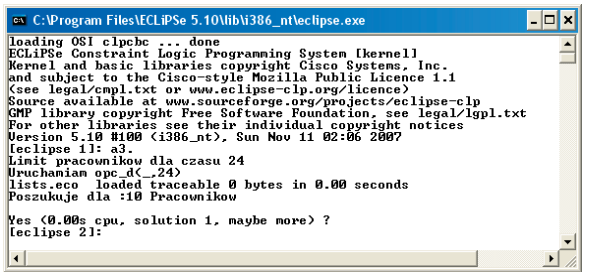


Rys. 3. Kolejność wykonywania zadań dla *przykład_1a* i *przykład_1b*

Dla powyższych danych wywołano następujące predykaty (*przykład_1a*): *opc_g(_,_)* (rys. 4) i *opc_d(_,24)* (rys. 5).



Rys. 4. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_g(_,_)* – wynik najkrótszy czas realizacji projektu = *24*



Rys. 5. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_d(_,24)* – wynik najmniejsza liczba pracowników dla czasu realizacji projektu = 24 L_{min}=*10*

Tab. 2. Harmonogramy dla *przykład_1a*

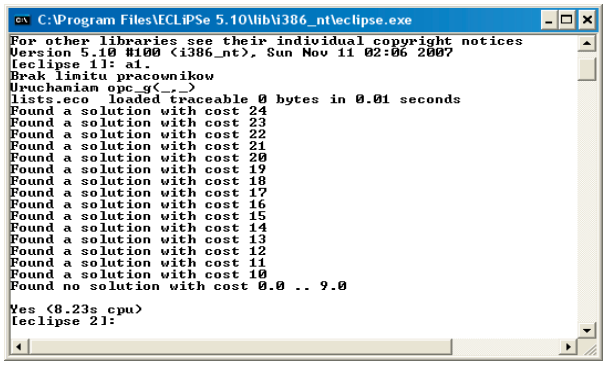
Operacja	*opc_g(_,_)* *Czas startu*	*opc_d(_,24)* *Czas startu*
A	0	0
B	2	2
C	5	5
D	5	5
E	8	8
F	10	10
G	13	13
H	17	17
I	17	22
J	22	24

Uzyskane wyniki (odpowiedzi na zadane pytania) pokazano na rys. 4 i 5. Odpowiadające im harmonogramy zadań w sposób tabelaryczny przedstawiono w tab. 2. Dane liczbowe dla *przykład_1b* przedstawiono w tab. 3. W odróżnieniu od poprzedniego przykładu czasy trwania poszczególnych zadań zależą od liczby przydzielonych dodatkowo pracowników. Przykładowo czas trwania zadania *A* wynosi 2 przy standardowej liczbie pracowników (tab. 1). Jeśli do wykonania zadania A przydzielony zostanie dodatkowy pracownik to czas realizacji zmniejszy się o 1. Przydział dodatkowych pracowników w tym przypadku nie zmieni tego czasu (tab. 3)

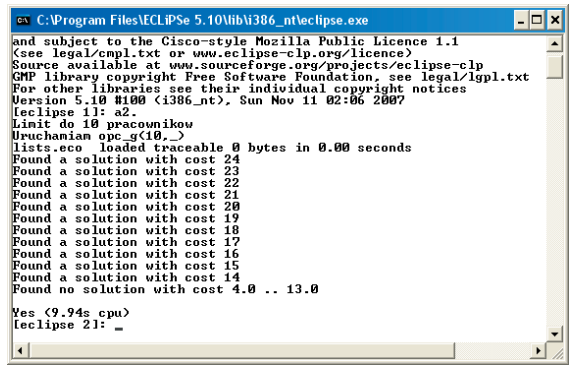
Tab. 3. Dane liczbowe *przykład_1b*

Operacja	Czas	Poprzed-nik	Dodatkowi pracownicy			
			0	**1**	**2**	**3**
A	2	-	0	1	1	1
B	3	A	0	0	1	2
C	4	B	0	1	2	3
D	3	B	0	1	1	2
E	2	D	0	1	1	1
F	3	C, E	0	1	2	2
G	4	F	0	1	1	2
H	5	G	0	1	2	3
I	3	G	0	1	1	2
J	2	H, I	0	1	1	1

Dla powyższych danych (*przykład_1b*) wywołano następujące predykaty: *opc_g(_,_)* (rys. 6) i *opc_g(10,_)* (rys. 7)



Rys. 6. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_g(_,_) –wynik najkrótszy czas realizacji projektu* =10



Rys. 7. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_g(10,_) – wynik najkrótszy czas realizacji projektu* =14

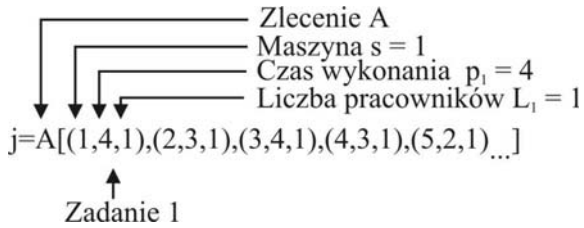
Tab. 4. Harmonogramy dla *przykład_1b*

Operacja	*opc_g(_,_) Czas startu*			*opc_g(1-,_) Czas startu*		
	Długość	Pracownicy	Start	Długość	Pracownicy	Start
A	1	5	0	1	5	0
B	1	11	1	2	10	1
C	2	6	2	3	5	3
D	1	6	2	2	4	3
E	1	4	3	1	4	5
F	1	8	4	1	8	6
G	2	11	5	3	9	7
H	2	7	7	2	7	11
I	2	7	7	1	9	10
J	1	9	9	1	9	13

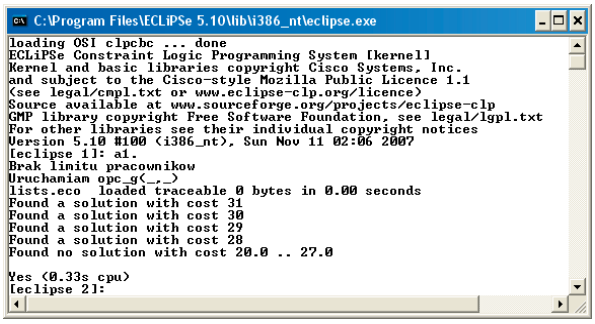
Drugim przykładem liczbowym jest harmonogramowanie w środowisku gniazdowym (job-shop) z uwzględnieniem dodatkowych zasobów (pracowników). W rozpatrywanym przykładzie (*przykład_2*) dany jest system składający się z 6 różnych maszyn, w którym jest wykonywane 5 zleceń. Każde ze zleceń składa się z 6 zadań. Kolejność wykonania zleceń jest dowolna natomiast kolejność zadań w zleceniu jest ustalona, co wynika z organizacji typu job-shop. Dane liczbowe przykładu oraz strukturę zlecenia przedstawiono w tabeli 5

Tab. 5. Dane liczbowe *przykład_2*

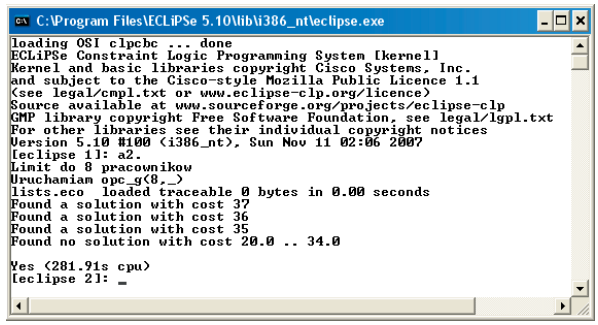
j∈{A,B,C,D,E}, o∈{1,2,3,4,5,6}, s∈{1,2,3,4,5,6}
j=A[(1,2,4), (2,4,2), (3,4,1), (4,2,1), (5,2,1), (6,3,4)]
j=B[(2,2,2), (3,3,3), (4,4,4), (1,2,3), (5,6,1), (6,3,2)]
j=C[(5,2,3), (4,2,1), (3,3,4), (2,4,3), (6,2,4), (1,4,4)]
j=D[(2,4,3), (3,2,6), (4,3,2), (5,2,3), (6,4,2), (1,4,4)]
j=E[(1,2,3), (3,4,6), (5,4,2), (6,4,2), (4,3,2), (2,2,2)]



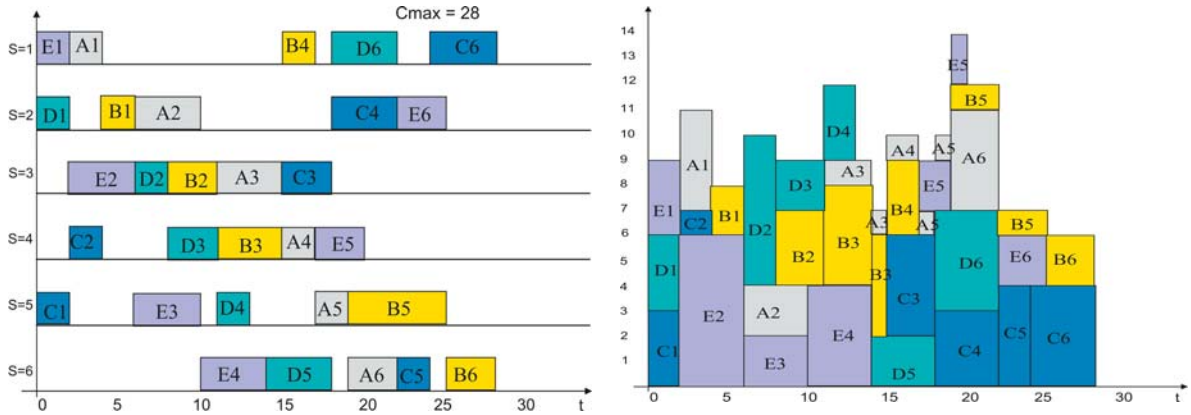
Dla powyższych danych (*przykład_2*) wywołano następujące predykaty: *opc_g(_,_)* (rys. 8, 10), *opc_g(8,_)* (rys. 9, 11), *opc_d(_,35)* (rys. 12), *opc_s(10,30)* (rys. 13), *opc_s(10,28)* (rys. 14).



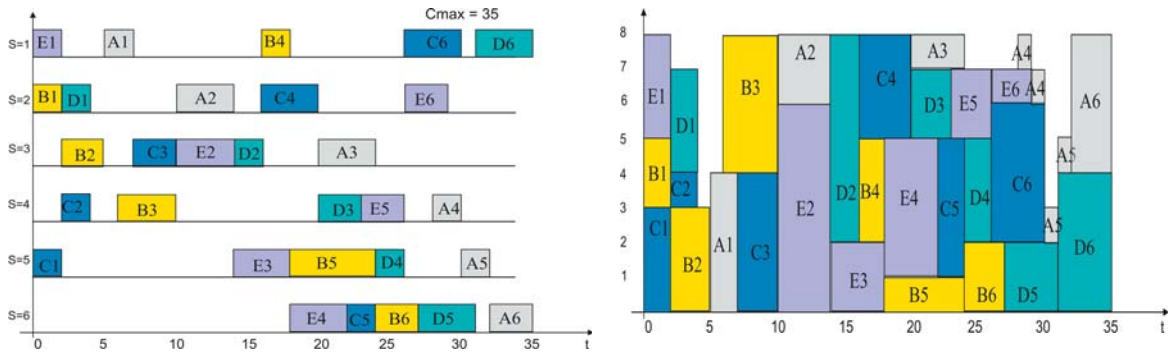
Rys. 8. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_g(_,_)*–wynik C^*_{max}=28, L=14



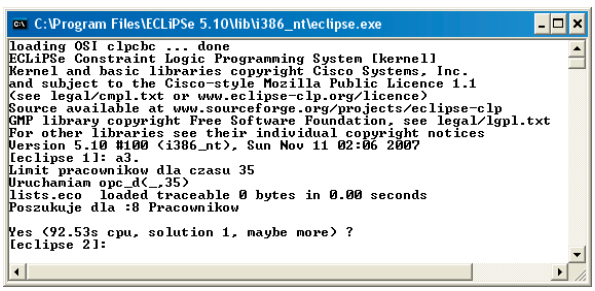
Rys. 9. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_g(8,_)*– wynik C^*_{max}=35,L=8



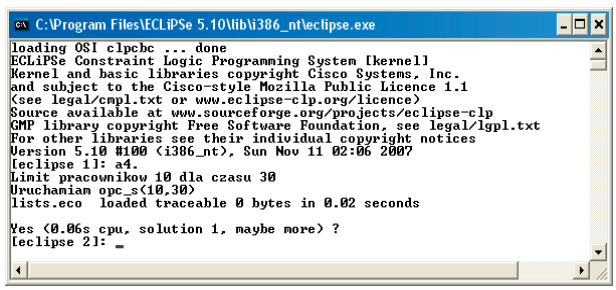
Rys. 10. Wykresy Gantt'a uzyskane na podstawie odpowiedzi na predykat *opc_g(_,_)*



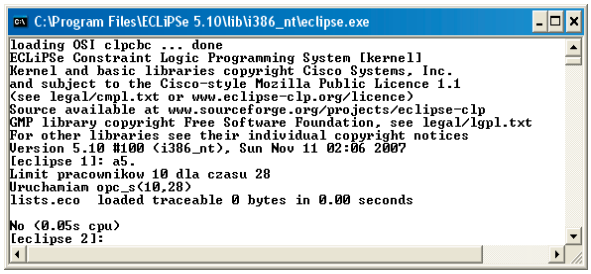
Rys. 11. Wykresy Gantt'a uzyskane na podstawie odpowiedzi na predykat *opc_g(8,_)*



Rys. 12. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_d(_,35)*–wynik, L_{min}=8



Rys. 13. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_s(10,30)*–wynik *Yes*



Rys. 14. Odpowiedź solvera Eclipse na pytanie zaimplementowane w predykacie *opc_s(10,28)*–wynik *No*

6. PODSUMOWANIE

Nawet pobieżna analiza uzyskanych wyników może wskazywać na duże możliwości zastosowań powyższej metodyki wspomagania decyzji w problemach harmonogramowania produkcji. Prezentowany system poprzez swoje uniwersalne struktury wpiera wszystkie formy organizacji produkcji od produkcji powtarzalnej o organizacji np. gniazdowej (job-shop) *przyklad_2* do produkcji unikalnej/jednostkowej –*przykład_1a* i *1b*. W systemie można również uwzględniać dodatkowe zasoby zewnętrzne (ludzie, specjalistyczne narzędzia, transport itp.) oraz czasową niedostępność maszyn lub/i zasobów dodatkowych. Powyższa funkcjonalność jest bardzo ważna dla przedsiębiorstw, w których ten typ zasobów jest krytyczny (np. MŚP). Można również sprawdzać wykonalność dodatkowych zadań w systemie już obciążonym wykonaniem podstawowego zbioru zadań. Przy wspomaganiu decyzji z wykorzystaniem powyższej metodyki oraz prezentowanego systemu można uwzględniać w podejmowanych decyzjach rozdział obciążeń. Można to robić na wiele sposobów np. uzależniając czas wykonania zadania od ilości przydzielonych zasobów (*przykład_1b*), od czasu startu lub innej dowolnej zależności. Dodatkowym atrybutem prezentowanych rozwiązań jest deklaratywność zastosowanych środowisk oraz ich wzajemna integracja. Środowiska deklaratywne umożliwiają, bowiem szybkie prototypownie i modelowanie problemu oraz dodawanie dowolnych ograniczeń wynikających np. ze specyfiki problemu lub wymagań klientów. Zaproponowana architektura (rys. 2) umożliwia implementację w środowiskach sieci Internet/Intranet/WAN i zdalną eksploatację systemu.

7. LITERATURA

1. JANIAK A.: *Wybrane problemy i algorytmy szeregowania zadań i rozdziału zasobów*. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999,
2. LIAO S. Y., WANG H. Q., LIAO L. J.: *An extended formalism to constraint logic programming for decision analysis*, Knowledge-based Systems 15, 2002 , pp. 189-202.
3. LEE H.G., LEE G. Yu., "Constraint logic programming for qualitative and quantitative constraint satisfaction problems", Decision Support Systems 16 (1), 1996, pp. 67-83.
4. M. Mouhoub, F. Charpilet, J. P. Haton : *Experimental Analysis of Numeric and Symbolic Constraint Satisfaction Techniques for temporal Reasoning*, Constraints, International Journal, 2, pp. 151-164, 1998.