Informatyka w Edukacji, XVI
UMK Toruń, 2019

SZEREGOWANIE ZADAŃ W ZROBOTYZOWANYM GNIEŹDZIE TECHNOLOGICZNYM, JAKO PRZYKŁAD MODELOWANIA ZŁOŻONYCH SYTUACJI PROBLEMOWYCH

Sławomir Herma, Włodzimierz Raczek, Bartłomiej Żywczak
V Liceum Ogólnokształcące, Bielsko-Biała ul. J.Lompy 10
Akademia Techniczno-Humanistyczna, ul. Willowa 2,
43-300 Bielsko-Biała
slawomir.herma@lo5.bielsko.pl, sherma@ath.bielsko.pl,
wlodzimierz.raczek@lo5.bielsko.pl,
bartlomiej.zywczak@lo5.bielsko.pl

Abstract. This paper presents a model of robotized work centers scheduling problem, using the multi-stage programming method. An analysis of its applicability for the educational purposes and the efficiency of algorithm was also described. On this basis there was made its computer implementation and evaluation of obtained solutions. The work describes the results of didactic experiments, conducted by the authors at various stages of academic and school education.

1 Wstęp

Podstawowym celem niniejszej pracy jest prezentacja efektów prowadzonych przez Autorów, na przestrzeni kilku ostatnich lat, eksperymentów dydaktycznych, dla których podmiotem badawczym jest szczególnie uzdolniona młodzież licealna oraz studenci wybranych kierunków studiów inżynierskich. Motywację stanowi chęć poszukiwania zagadnień dających nie tylko przestrzeń wdrażania kompetencji kluczowych w jak najszerszym ujęciu lecz przede wszystkim wykazujących duży potencjał dydaktyczny, pozwalających elastycznie stopniować poziom trudności, umożliwiających nauczycielowi (również akademickiemu) swobodnie kształtować określone aktywności podczas zajęć i osiągać zaplanowane efekty. Wydaje się, że do rodziny takich zagadnień można dołączyć obszerny zestaw problemów decyzyjnych oraz optymalizacyjnych podejmujących temat szeregowania zadań w systemach wytwarzania.

Zagadnienie harmonogramowania pracy zrobotyzowanych gniazd produkcyjnych należy problemów o charakterze operacyjnym, które choć doczekały się szeregu opracowań [1, 2, 4] prezentujących rozwiązania z zakresu algorytmiki, sztucznej inteligencji czy badań operacyjnych, pozostają w dalszym ciągu otwarte na szereg usprawnień, modyfikacji oraz przykładów praktycznego zastosowania.[3] Niniejszy artykuł podejmuje więc próbę przedstawienia tej problematyki w kontekście budowy przystępnych rozwiązań numerycznych, kształtujących szereg intuicji podczas tworzenia gier dydaktycznych, modeli symulacyjnych lub programów komputerowych, co wydaje się sprzyjać oczekiwaniom zwłaszcza ze strony ambitnej młodzieży.

2 Model zagadnienia

2.1 Dane wejściowe

Niech będzie dane pojedyncze gniazdo produkcyjne, realizujące jedną lub kilka operacji (np. montażowych) na pewnym zbiorze przedmiotów (elementów, części podzespołów, modułów itp.), w ramach pewnego procesu technologicznego. Obsługa (przetwarzanie) każdego elementu wykonywana jest szeregowo tj. w czasie liczonym od chwili jego dostępności, aż do momentu opuszczenia gniazda

Niech będzie dany zbiór elementów podlegających obsłudze:

$$\Omega = \left\{ \omega_n \right\}_{n=1,\ldots,N}$$

gdzie n oznacza liczbę tych elementów, a N - numer ostatniego elementu zbioru. Dla każdego elementu znany jest czas obsługi (mierzony w dowolnie przyjętych jednostkach np. [s]), podany w postaci wektora:

$$\Theta = \left[\vartheta_n \right]_{n=1,\ldots,N}$$

Ponieważ gniazdo produkcyjne jest jednym z wielu składowych systemu wytwarzania, jest więc determinowane harmonogramem swych poprzedników. Stąd poszczególne elementy zbioru Ω posiadają różne chwile dostępności wynikające z różnych chwil zakończenia operacji technologicznych realizowanych na wcześniejszych etapach procesu wytwarzania. Oznacza to, że rozpoczęcie obsługi wybranego elementu w badanym gnieździe nie może nastąpić wcześniej, niż określa to jego dostępność. Wektor chwil dostępności poszczególnych elementów jest więc kolejnym źródłem danych wejściowych w opisywanym modelu:

$$\Phi = \left[\varphi_n \right]_{n=1,\ldots,N}$$

Podobnie, realizacja dalszych czynności wynikających z przyjętej w systemie wytwarzania marszruty technologicznej, wymusza konieczność „wypuszczenia” obsłużo-

nych już elementów z badanego gniazda w określonych chwilach. Są to chwile deklarowane, przechowywane w kolejnym wektorze:

$$\Psi = \left[\psi_n\right]_{n=1,\ldots,N}$$

Jeżeli definiować pojęcie harmonogramu jako ogół procedur organizacyjnych, zmierzających do ustalenia zależności czasowych bądź czasoprzestrzennych dla układów typu „obiekt – procesor", podstawowym celem przedstawianego modelu jest znalezienie takiej ścieżki kolejności obsługi poszczególnych elementów, by różnice pomiędzy faktycznym terminem zakończenia, a zadeklarowanym – dla każdego elementu były jak najmniejsze (oczywiście przy uwzględnieniu różnych chwil dostępności na wejściu układu).

2.2 Ograniczenia modelu (warunki brzegowe)

Niezależnie od przyjętego sposobu rozwiązania, każdy uzyskany harmonogram podlega ocenie dopuszczalności. Musi więc spełniać następujące ograniczenia:

- w dowolnej chwili w badanym gnieździe może znajdować się tylko jeden element:

$$\underset{i}{\forall}\ \underset{j\neq i}{\forall}\left(t_i \le \rho_j\right) \vee \left(t_j \le \rho_i\right)$$

gdzie: t_i - chwila zakończenia obsługi elementu ω_i

ρ_i - chwila rozpoczęcia obsługi elementu ω_i

- nie można rozpocząć obsługi elementu zanim nie stanie się on dostępny na wejściu badanego układu:

$$\underset{1\le n\le N}{\forall}\ \rho_n \ge \varphi_n$$

2.3 Kryterium optymalności

Uzyskanie zbioru harmonogramów dopuszczalnych stanowi podstawę poszukiwania rozwiązania optymalnego. W opisywanym modelu kryterium polega na minimalizacji maksymalnego opóźnienia obsługi elementów w gnieździe tzn.:

$$Q = \max_{1\le n\le N}\left(t_n - \psi_n\right) \to \min$$

Opóźnienie to jest różnicą pomiędzy rzeczywistą, a deklarowaną chwilą zakończenia obsługi każdego elementu.

2.4 Harmonogram pracy gniazda

Harmonogram pracy gniazda to uporządkowany ciąg par chwil rozpoczęcia i zakończenia obsługi każdego elementu zbioru Ω:

$$H = \left\langle \left\langle \rho_1, t_1 \right\rangle, \left\langle \rho_2, t_2 \right\rangle, ..., \left\langle \rho_n, t_n \right\rangle, ..., \left\langle \rho_N, t_N \right\rangle \right\rangle$$

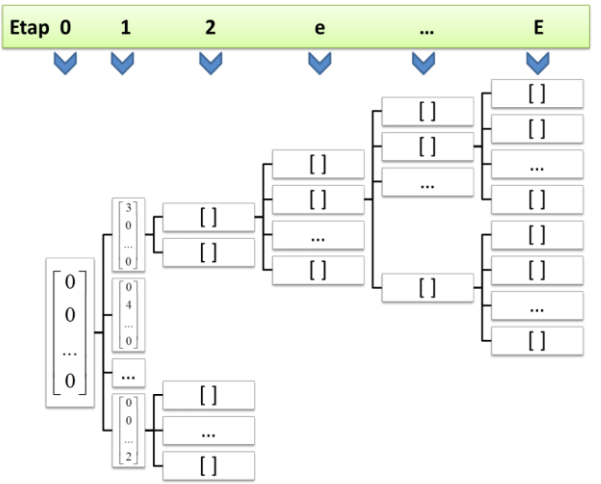
Chwila zakończenia obsługi n-tego elementu jest wyznaczana jako suma chwili ρ_n i czasu trwania obsługi ϑ_n:

$$t_n = \rho_n + \vartheta_n$$

Zatem harmonogram można zapisać w uproszczonej postaci, jako ciąg chwil zakończenia obsługi:

$$H = \left\langle t_1, t_2, ..., t_n, ..., t_N \right\rangle$$

Zgodnie z propozycją przedstawioną przez Mareckiego w [2], do rozwiązania opisywanego zagadnienia przyjęto model programowania wieloetapowego, polegający na permutacyjnym generowaniu przyporządkowań elementów ω_n, a następnie wyborze optymalnego spośród uzyskanych rozwiązań.



Rysunek 1 Struktura drzewa procesu decyzyjnego

Realizacja tego zadania wymaga więc użycia jednowymiarowej struktury tablicowej (wektora) do zdefiniowania pojedynczego stanu procesu decyzyjnego:

$$P^{e,l} = \left[p_i^{e,l} \right]_{i=1,\ldots,N}$$

gdzie:

$$p_i^{e,l} = \begin{cases} t_i, \text{ gdy } \omega_i \text{ wpisany do harmonogramu} \\ 0, \text{ w przeciwnym wypadku} \end{cases}$$

oraz:

e - numer etapu, na którym podjęto decyzję

l - numer kolejny wektora stanu na danym etapie.

2.5 Generowanie drzewa decyzyjnego

W ujęciu deterministycznym, utworzenie kolejnego wektora w strukturze drzewa procesu decyzyjnego odbywa się zgodnie następującą procedurą::

$$\underset{1 \le n \le N}{\forall} \left(p_n^{e-1,\lambda} = 0 \right) \Rightarrow \left(P^{e,l} = P^{e-1,\lambda} + \Delta p \right)$$

gdzie:

$$\Delta p_i = \begin{cases} t_n, i = n \\ 0, i \neq n \end{cases}$$

oraz:

$$t_n = \rho_n + \vartheta_n$$

i:

$$\rho_n = \max\left(\varphi_n, T^{e-1,\lambda} \right)$$

Decyzję o wprowadzeniu do harmonogramu pracy gniazda kolejnego elementu można podjąć dla każdego $\omega_n \in \Omega$, dla którego w wektorze stanu, na pozycji n znajduje się wartość „0" (tzn. element ten do tej pory nie został jeszcze obsłużony). Wówczas kolejny stan procesu decyzyjnego $P^{e,l}$ (o numerze *l*, na etapie *e*) wyznacza się jako suma wektora stanu poprzedniego $P^{e-1,\lambda}$ i pewnego wektora Δp.

Wektor Δp ma taki sam wymiar co wektor stanu, i tylko na tej pozycji, która odpowiada przydzielanemu elementowi, posiada wartość niezerową, równą rzeczywistej chwili zakończenia obsługi t_n elementu. Wartość t_n można łatwo wyznaczyć, dodając do chwili rzeczywistego rozpoczęcia obsługi elementu ρ_n czas jego obsługi ϑ_n,

zawarty w zestawie danych wejściowych. Zgodnie z przyjętym na początku założeniem, że dopóki nie zakończy się obsługa jednego elementu nie wolno przydzielić następnego, wyznaczanie chwil rozpoczęcia ρ_n jest zdeterminowane zarówno dostępnością elementu ω_n – czyli chwilą φ_n, jak i rzeczywistą chwilą zakończenia obsługi ostatniego dotychczas przydzielonego elementu – czyli chwilą $T^{e-1,\lambda}$. Z tych dwóch wartości należy zatem wybrać chwilę późniejszą i właśnie ją przyjąć jako ρ_n.

2.6 Możliwości rozszerzania modelu

W praktyce, funkcjonowanie gniazda produkcyjnego podlega pewnym ograniczeniom związanym np. z koniecznością zapewnienia ściśle określonej kolejności dla pewnych sekwencji operacji (lub elementów). W celu uwzględnienia tego faktu, należy zestaw danych wejściowych uzupełnić o tzw. macierz poprzedników i następników, zgodnie z następującą definicją:

$$\Gamma = \left[\gamma_{\nu,n} \right]_{\nu,n=1,\ldots,N}$$

gdzie:

$$\gamma_{\nu,n} = \begin{cases} 1, \omega_\nu \Rightarrow \omega_n \\ 0, \text{ w przeciwnym wypadku} \end{cases}$$

Choć informatyczna implementacja tej struktury nie wydaje się uciążliwa, należy jednak wziąć pod uwagę konieczność zbudowania mechanizmów programowych weryfikujących niesprzeczność informacji tu zawartych. Jeśli założyć dodatkowo, że pomiędzy elementami (lub operacjami) ω_n istniejące zależności wzajemnego następstwa mogą mieć zarówno bezpośredni, jak i pośredni charakter, problem właściwego zaprogramowania i obsługi macierzy kolejności, do zagadnień trywialnych już nie należy. Odrębnego komentarza wymaga również fakt, że użytkownik – wypełniając macierz poprzedników i następników, wpływa na efektywność procesu przeszukiwania obszaru możliwych rozwiązań harmonogramu. W im większym stopniu kolejność obsługi elementów zostanie zdeterminowana, tym mniej liczny będzie zbiór harmonogramów dopuszczalnych. W szczególności, może się okazać, że zaistnieje tylko jedna ścieżka przebiegu procesu decyzyjnego , skutkująca zanikiem samego zadania optymalizacyjnego.

Pozostałe dane wejściowe modelu nie ulegają zmianie, modyfikacji natomiast ulega procedura generowania stanów, która po uwzględnieniu zależności kolejnościowych przyjmie następującą postać:

$$\underset{1\le n\le N}{\forall}\ \underset{1\le v\le N}{\forall}\left\{\left(p_n^{e-1,\lambda}=0\right)\wedge\left[\left(\gamma_{v,n}=1\right)\Rightarrow\left(p_v^{e-1,\lambda}\neq 0\right)\right]\right\}\Rightarrow\left(P^{e,l}=P^{e-1,\lambda}+\Delta p\right)$$

Kolejnym uzupełnieniem modelu, wynikającym z konieczności uwzględnienia dodatkowego czasu na dostosowanie gniazda do warunków wymaganych wytwarzaniem określonego elementu, jest tzw. czas przezbrojenia. Jest on zależny nie tylko od rodzaju elementu, który ma zostać obsłużony, lecz także od tego, którego obsługa właśnie się zakończyła. Stąd, w opisywanym modelu harmonogramowania, w zestawie jego danych wejściowych, należy dodatkowo zdefiniować tzw. macierz czasów przezbrojeń jako:

$$\mathrm{T}=[\tau_{i,j}]_{i,j=1,\ldots,N}$$

gdzie:

$\tau_{i,j}$ - jest czasem przezbrojenia gniazda między obsługą elementu ω_i i ω_j,

przy czym:

$$\tau_{i,j}\neq\tau_{j,i}$$

Wprowadzenie dodatkowego parametru wejściowego nie zmienia zasadniczo dotychczasowego modelu. Poszukiwanie rozwiązań dopuszczalnych nadal przebiega zgodnie metodą programowania wieloetapowego i opisaną procedurą generowania stanów. Jednakże definicje niektórych wielkości ulegają nieznacznej modyfikacji – w szczególności formuła określająca chwilę rozpoczęcia obsługi elementów:

$$\rho_n=\max\left(\varphi_n,T^{e-1,\lambda}+\tau_{K^{e-1,\lambda},n}\right)$$

gdzie:

$K^{e-1,\lambda}$ - jest numerem elementu który został obsłużony jako ostatni na etapie (e-1), i wyznaczany jest jako:

$$K^{e-1,\lambda}=\begin{cases}j,\ p_j^{e-1,\lambda}=T^{e-1,\lambda}\\ 0,\ \text{w przeciwnym wypadku}\end{cases}$$

Choć chwila zakończenia obsługi elementu uwzględnia czas przezbrojenia, zawsze jednak należy zbadać, która z dwóch wartości jest większa (czy chwila dostępności nowego elementu, czy suma chwili zakończenia i czasu przezbrojenia dla elementu poprzedniego) i tę należy przyjąć jako moment rozpoczęcia jego obsługi. Istnienie czasów przezbrojeń jest zjawiskiem powszechnym i naturalnym w uwarunkowaniach

produkcyjnych większości przedsiębiorstw. Dążenie więc do znalezienia takich marszrut obsługi elementów, w których międzyoperacyjne czasy oczekiwania na rozpoczęcie pracy gniazda byłyby jak najmniejsze, wydaje się oczywiste.

Oprócz powyższych rozszerzeń zasadne jest ponadto zwrócenie uwagi na możliwe, dodatkowe kryteria oceny znalezionych rozwiązań:

– kryterium minimalizacji spóźnienia na wejściu gniazda:

$$Q_2 = \max_{1 \le n \le N} (\rho_n - \varphi_n) \to \min$$

– kryterium minimalizacji kary za spóźnienie rozpoczęcia obsługi:

$$Q_3 = \sum_{1 \le n \le N} c_n \cdot (\rho_n - \varphi_n) \to \min$$

W tym przypadku należy zestaw danych wejściowych uzupełnić o dodatkowy wektor zawierający stawki kar za jednostkę czasu powstałego opóźnienia:

$$C = [c_n]_{n=1,\ldots,N}$$

– kryterium minimalizacji kary za spóźnienie na wyjściu układu:

$$Q_4 = \sum_{1 \le n \le N} c_n \cdot (t_n - \psi_n) \to \min$$

– kryterium minimalizacji czasu oczekiwania na rozpoczęcie obsługi :

$$Q_5 = \sum_{i=1}^{N} q_i \to \min$$

gdzie: $q_i = \rho_i - T^{e-1,\lambda} \ge 0$

Nie ogranicza to jednak możliwości tworzenia dalszych kryteriów, uwzględniających jedocześnie kilka z pokazanych wyżej sposobów oceny uzyskanych harmonogramów. Można dzięki temu poszukiwać rozwiązań optymalnych w sensie Pareto – użytecznych zwłaszcza w wielu zagadnieniach z zakresu inżynierii produkcji.

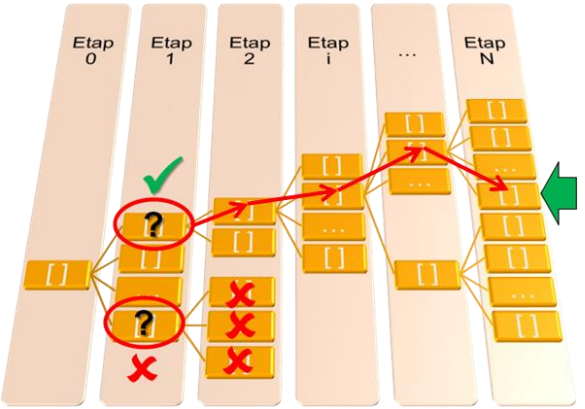
2.7 Ocena i optymalizacja złożoności algorytmu

Zagadnienie harmonogramowania (szeregowania) zadań w układzie gniazda produkcyjnego, realizowane metodą programowania wieloetapowego należy do zagadnień NP-trudnych, dla których obserwuje się bardzo szybki wzrostu struktury drzewa

decyzyjnego. Ograniczenie liczby generowanych wektorów stanu, a tym samym znaczne przyspieszenie działania algorytmu można poprawić:

- poprzez wprowadzenie wspomnianych rozszerzeń np. zależności kolejnościowych obsługi elementów, co spowoduje znaczne zmniejszenie liczby generowanych permutacji i bezpośrednio przełoży się na efektywność czasową oraz pamięciową procesu;
- poprzez wykorzystanie heurystyk eliminacji nieperspektywicznych stanów decyzyjnych (w przypadku harmonogramowania gniazda – tzw. reguły sondowania).

Regułę sondowania można zastosować na każdym etapie procesu decyzyjnego. W ogólności polega ona na ocenie każdego wektora stanu na bieżącym etapie według przyjętego kryterium, a następnie zakwalifikowaniu go do grupy perspektywicznych bądź całkowitemu zdyskwalifikowaniu. Pod pojęciem stanu perspektywicznego rozumie się taki, którego osiągnięta na danym etapie wartość daje potencjalną szansę uzyskania rozwiązania optymalnego na końcu procesu decyzyjnego, a tym samym stanowić podstawę generowania kolejnych ścieżek decyzyjnych.



Rysunek 2 Eliminacja nieperspektywicznych stanów decyzyjnych

Tok postępowania przy stosowaniu reguły sondowania jest następujący:

- na wybranym, bieżącym etapie e należy zidentyfikować dwa dowolne wektory stanu o numerach l_1 i l_2, przeznaczone do dalszej analizy. Wyznaczone według nich zbiory dotychczas obsłużonych elementów mogą być różne.
- dla obu wektorów należy wyznaczyć chwilę zakończenia obsługi ostatniego z przydzielonych elementów :

$$P^{e,l_1} \rightarrow T^{e,l_1} \text{ oraz } P^{e,l_2} \rightarrow T^{e,l_2}$$

- przydzielić dotąd nie przydzielone elementy ze zbioru Ω, uzyskując dwa „prognozowane" wektory końcowe. Należy uwzględnić chwile dostępności elementów.
- Wyznaczyć i porównać oba stany. Uznać za nieperspektywiczny ten, którego ocena z punktu widzenia przyjętego kryterium jest niższa.

3 Propozycje rozwiązań zagadnienia

3.1 Model symulacyjny i gra decyzyjna w MsExcel

Zaprezentowany model problemu szeregowania zadań korzysta ze struktur danych, które po przeniesieniu do arkusza kalkulacyjnego np. MsExcel lub analogicznego, dają podstawę utworzenia interesującej przestrzeni symulacyjnej. Bez konieczności stosowania języka VBA lub innego (np. Open Office Basic, Libre Office Basic) uczniowie (lub studenci) mają możliwość zbudowania arkusza, w którym rozwiązanie problemu będzie odbywać się wg algorytmu zachłannego, zaproponowanego jako tzw. reguła sondowania. Przedstawienie zadania jako **gry decyzyjnej** okazuje się kluczowe z punktu widzenia atrakcyjności prowadzonych zajęć. Ich ramowy scenariusz mógłby przedstawiać się następująco:

- Prezentacja zagadnienia, wyjaśnienie istoty problemu, istniejących ograniczeń oraz koniecznych do uwzględnienia danych wejściowych;
- Utworzenie wstępnego arkusza, przyjęcie liczby elementów do obróbki w gnieździe, określenie czasów obsługi, najwcześniejszych możliwych chwil dostępności, najpóźniejszych dopuszczalnych chwil zakończenia, Żmudne prowadzanie danych numerycznych można przyspieszyć, wykorzystując funkcję LOS lub LOS.ZAKR (MsExcel).
- Przyjęcie ograniczeń kolejnościowych (w macierzy poprzedników i następników). Zwracanie uwagi, by podawane wielkości nie ograniczyły zbytnio możliwości prowadzenia procesu decyzyjnego w dalszej części zajęć.
- Wyodrębnienie zakresu komórek (np. jednej z kolumn) z przeznaczeniem na wektor stanu, w którym grający będą mieć możliwość decydowania o kolejności obsługi elementów w gnieździe. Obok, za pomocą wbudowanej funkcji INDEKS należy zadbać o prawidłowe pobieranie wartości danych wejściowych .
- Zbudowanie dynamicznego wykresu Gantt-a (dającego możliwość wizualizacji rezultatów prowadzonych symulacji). Kontrola wyników i wybór najlepszych rozwiązań.

	A	B	C	D	E	F	G	H	I	J	K	L	M
2	DANE WEJŚCIOWE					WYNIKI SYMULACJI							
3	nr op.	teta	phi	psi		sekwencja zadań	teta	phi	psi	pocz	koniec	spóźnienie na wyjściu	oczekiwanie na wejściu
4	1	5	3	55		3	2	7	15	0	9	-6	-7
5	2	7	5	20		6	7	7	30	9	16	-14	2
6	3	2	7	15		5	8	9	17	16	24	7	7
7	4	3	3	27		2	7	5	20	24	31	11	19
8	5	8	9	17		7	4	8	14	31	35	21	23
9	6	7	7	30		8	6	3	25	35	41	16	32
10	7	4	8	14		4	3	3	27	41	44	17	38
11	8	6	3	25		9	2	5	35	44	46	11	39
12	9	2	5	35		1	5	3	55	46	51	-4	43
13	10	8	7	40		10	8	7	40	51	59	19	44

Rysunek 3 Przykład arkusza MsExcel z projektem gry decyzyjnej

Nieco trudniejszym zadaniem jest budowa arkusza symulacyjnego uwzględniającego wszystkie opisane w modelu dane wejściowe i ograniczenia – w szczególności macierz poprzedników i następników. Zaprogramowanie mechanizmów umożliwiających automatyczną kontrolę niesprzeczności tej macierzy (zarówno dla wariantu zależności pośrednich i bezpośrednich pomiędzy operacjami) wymaga już umiejętności posługiwania się językiem VBA. Zadaniem tym można obarczyć szczególnie uzdolnionych uczniów lub studentów, podobnie, jak budową aplikacji według algorytmu deterministycznego z uwzględnieniem reguły sondowania.

Interesującą alternatywą wydaje się być również skorzystanie z dodatku SOLVER system MsExcel i porównanie efektów jego działania zależnie od wariantu wybranej metody optymalizacyjnej.

3.2 Aplikacja komputerowa

Istotnym elementem prowadzonego eksperymentu dydaktycznego było zbudowanie i przetestowanie aplikacji komputerowej, napisanej w środowisku Embarcadero Delphi. Zadanie to zostało wykonane przez studentów kierunków informatycznych w ramach prac dyplomowych. Przykładowy interfejs użytkownika przedstawia rys. 4.

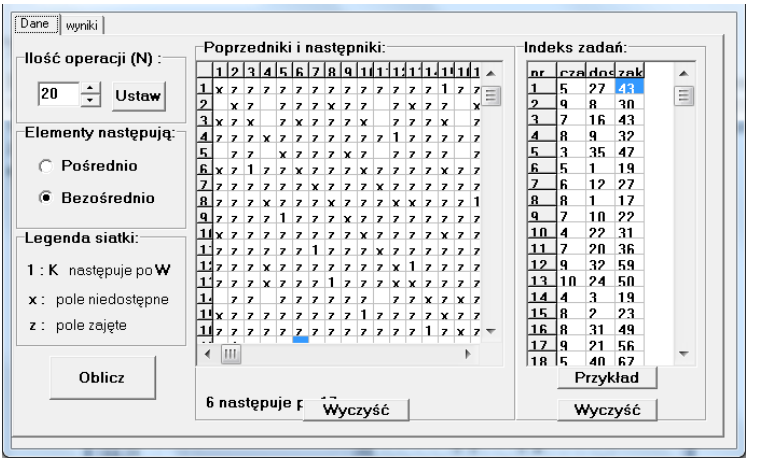
Implementacja opisanego modelu umożliwiła przeprowadzenie symulacji procesu wytwarzania zarówno w gniazdach produkcyjnych przedmiotowych, jak i technologicznych. Choć wykorzystanie metody programowania wieloetapowego powodowało tworzenie bardzo nieraz rozległych drzew decyzyjnych (co skutkowało obniżeniem efektywności czasowej aplikacji), udało się zmniejszyć złożoność obliczeniową i pamięciową wprowadzając odpowiednie restrykcje kolejnościowe (wynikające z istnienia macierzy poprzedników i następników) oraz uwzględniając regułę sondowania.

4 Podsumowanie

Problem szeregowania zadań w gnieździe produkcyjnym jest zagadnieniem modelowym, będącym uogólnieniem wielu sytuacji z którymi spotkać się można w życiu

codziennym. Można go uznać za klasę abstrakcji dla takich zadań jak organizacja pracy przychodni lekarskiej, gabinetu dentystycznego, salonu fryzjerskiego oraz większości form ludzkiej działalności, w których występują ograniczenia czasowe, kolejnościowe oraz potrzeba podejmowania decyzji wg określonych kryteriów optymalności.

W kontekście pracy uczniów, studentów oraz nauczycieli, realizacja zadań programistycznych tej klasy przynosi szereg korzyści, z których za najważniejsze należy uznać: zdobywanie doświadczenia w prowadzeniu projektów zespołowych, praktyczne stosowanie metod aktywizujących oraz wdrażanie kompetencji kluczowych.



Rysunek 4 Interfejs aplikacji wspomagającej harmonogramowanie gniazda produkcyjnego

Literatura

1. Banaszak Z., Bocewicz G.: *Decision Support Driven Models and Algorithms of Artificial Intelligence*, Warszawa, 2011.
2. Bucki R., Marecki F.: *Digital Simulation of Discrete Processes*, Network Integrators Associates, 2006.
3. Herma S., Raczek W., Żywczak B. *Problem komiwojażera w świetle kształtowania myślenia komputacyjnego na wybranych etapach edukacyjnych*, Informatyka w Edukacji, Wydawnictwo Naukowe UMK, Toruń 2018
4. Janiak A.: *Wybrane problemy i algorytmy szeregowania zadań i rozdziału zasobów*, Akademicka Oficyna Wydawnicza PLJ, 1999.