Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Alkalmazott Informatikai Intézeti Tanszék

Szakdolgozat

Erőforrás-korlátos projektütemezési feladatok eredményeit megjelenítő WEB-es alkalmazás fejlesztése

Készítette:

Név: Nagy Bence

Neptunkód: FVIQLY

Szak: mérnökinformatikus BSc

Termelésinformatika sáv

Miskolc, 2024

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar

Alkalmazott Informatikai
Intézeti Tanszék
3515 Miskolc-Egyetemváros

*Szak**: Mérnökinformatikus***

Azonosító szám:
GEIAK-FVIQLY/2024-25-1f/BSc

Specializáció/sáv: Termelésinformatika

Intézményazonosító: FI 87515

SZAKDOLGOZAT FELADAT
Nagy Bence

Neptun-kód: FVIQLY
BSc mérnökinformatikus jelölt részére

A tervezés tárgyköre: Termelésinformatika

A feladat címe: Erőforrás-korlátos projektütemezési feladatok eredményeit megjelenítő WEB-es alkalmazás fejlesztése

A feladat részletezése:

1. A szakdolgozat célja egy eredmény-menedzselő WEB-es alkalmazás kifejlesztése, amely grafikus felhasználói felület segítségével támogatja erőforrás-korlátos projektütemezési feladatok eredményeinek megjelenítését.
2. Szakirodalmi elemzés alapján vizsgálja meg az ütemtervek megjelenítésének és értékelésének lehetőségeit!
3. Dolgozzon ki alkalmas adatmodelleket, algoritmusokat és grafikus felhasználói felületeket a következő funkciók megoldására:
    a. Projektek és ütemtervek (megoldások) betöltését támogató bemeneti források kezelése.
    b. Az erőforrások rendelkezésre állásának betöltése és megjelenítése.
    c. A projektek végrehajtási ütemterveinek betöltése, feldolgozása és megjelenítése.
    d. A projektek és tevékenységek jellemzőinek (pl. végrehajtási idő, határidő, prioritás, előfeltételek stb.) megjelenítése.
    e. A végrehajtási tervek (ütemtervek) megjelenítése.
    f. Az ütemezés teljesítménymutatóinak számítása és megjelenítése.
4. Célszerűen választott technológiák és fejlesztőeszközök segítségével implementálja a fenti funkciókat megvalósító WEB-es alkalmazást!
5. Dolgozzon ki integrációs megoldást, amelyek segítségével a szoftver összekapcsolható létező ütemező motorral!
6. Ismertesse a kidolgozott szoftver felhasználói felületét és használatát!

Tervezésvezető: Dr. Kulcsár Gyula — egyetemi docens, Alkalmazott Informatikai Intézeti Tanszék

Konzulens:

A feladat kiadásának időpontja: 2024.09.09.

A feladat beadásának határideje: 2024.11.15.

Miskolc, 2024.09.09.

Dr. Nehéz Károly

tanszékvezető, egyetemi docens

1. *A szakmai gyakorlat helye:*

2. *A szakmai gyakorlat vezetőjének neve:*

3. *A szakdolgozat módosítása:* *szükséges (a módosítást külön lap tartalmazza)*
   nem szükséges (a megfelelő rész aláhúzandó)

   Miskolc, _____

   tervezésvezető aláírása

4. *A tervezést ellenőriztem:* *(1)* _____

   (2) _____

   (3) _____

   (4) _____

   dátum, tervezésvezető aláírása

5. *A szakdolgozat* *beadható*

   nem adható be

   Miskolc, _____

   konzulens aláírása *tervezésvezető aláírása*

6. *A szakdolgozat*
   ... szövegoldalt,
   ... db ábrát,
   ... db adathordozó mellékletet
   ...egyéb mellékletet tartalmaz.

7. *A szakdolgozat bírálatra:* *bocsátható*

   nem bocsátható

   A bíráló neve, címe: _____

   Miskolc, _____

   tanszékvezető aláírása

8. *Osztályzat:* *a bíráló javaslata: _____*

   a tanszék javaslata: _____

   a Záróvizsga Bizottság döntése: _____

   Miskolc, _____

   a Záróvizsga Bizottság elnökének aláírása

EREDETISÉGI NYILATKOZAT

Alulírott Nagy Bence; Neptun-kód:FVIQLY
a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős mérnökinformatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy Erőforrás-korlátos projektütemezési feladatok eredményeit megjelenítő WEB-es alkalmazás fejlesztése című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom
felhasználása a forráskezelés szabályai szerint történt.
Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,.....év ……..hó ……..nap

…….……..…
Hallgató

Tartalom

1. Bevezetés

A modern gazdaságban a termelés és a szolgáltatás különféle erőforrásokat igényel, amelyek csak korlátozottan állnak rendelkezésre. Ezen korlátozott erőforrások hatékony kihasználása kulcsfontosságú a rendszerek zavartalan működéséhez, amihez ütemező szoftverek nyújtanak nélkülözhetetlen támogatást. Az ütemezési feladatok a konkrét problémák széles skáláját foglalják magukba, amelyek közül kiemelkedik a projektütemezés, amely komplex döntéshozatali folyamatokat és pontos tervezést igényel.

A szakdolgozat célja az erőforrás-korlátos projektütemezési feladat (Resource-Constrained Project Scheduling Problem, RCPSP) megoldásának támogatása egy kliens-szerver alapú megközelítéssel. A dolgozathoz tartozó szoftverfejlesztés során a teljes kliens oldali fejlesztés valósul meg, amely biztosítja a felhasználói interakciót és a szerverrel való kapcsolattartást. Ezen felül a fejlesztés részét képezi a szerver oldali kommunikációs módszer kialakítása, valamint a szerver oldalhoz érkező bemeneti adatok ellenőrzése, amely elengedhetetlen a megbízható működéshez. Az ütemező algoritmus fejlesztése nem része a feladatnak. A szoftverprojektem egy meglévő C++ alapú implementációt integrál a szerver oldalára, így lehetőséget nyújt a már jól működő ütemező algoritmus hasznosítására a tervezett rendszerben.

A szakdolgozat lépésről lépésre mutatja be a projektütemezési feladatok elméleti alapjait, a fejlesztett rendszer struktúráját és annak gyakorlati megvalósítását. A második fejezet az alapfogalmakat tárgyalja, amelyek szükségesek a projekt- és termelésütemezési környezet megértéséhez. Ez a rész ismerteti a termelés és munka fogalmát, a precedencia gráf használatát, valamint az erőforrás- és ütemezési alapokat, beleértve a Gantt diagramot és a projekt fogalmát. Ezt követően a termelés- és projektütemezés elméletére tér ki, bemutatva a termelésinformatika szerepét és a számítógéppel integrált gyártás koncepcióját, valamint a webes alkalmazások struktúráját.

A harmadik fejezet a fejlesztendő rendszer alapmodelljét ismerteti, különös tekintettel a kliens-szerver koncepcióra, a kommunikációs módszerek és a szerver rugalmasságának biztosítására.

A negyedik fejezet részletesen bemutatja a főbb szoftvermodulokat, beleértve az ütemezést megoldó szerver, a kliens és a felhőalapú alkalmazás karbantartó moduljait. Emellett ez a fejezet tárgyalja a szoftvertervezés során kialakult legfontosabb osztályokat.

Az ötödik fejezet a megvalósítás különböző kísérleteit és kihívásait mutatja be, mint például a modulok DLL-ként való fordítását, a cpprest könyvtár alkalmazását, valamint a kliens-oldali kommunikációs osztály JavaScript-ben való megvalósítását. Ez a rész kitér a felhőben történő elhelyezés és az nginx alapú átirányítás technikai részleteire is.

A hatodik fejezet a kész rendszer bemutatását tartalmazza, beleértve a konténerizáció részleteit, a szoftver használati eset diagramját és a kulcsfontosságú kódrészleteket mind a szerver, mind a kliens oldalról.

A szakdolgozat lehetővé teszi az olvasó számára, hogy átfogó képet kapjon a választott témámról, az elvégzett modellezési, tervezési, szoftverfejlesztési és tesztelési munkámról valamint a megoldás részleteiről.

2. Elméleti háttér

2.1 Termelés

2.1.1 A termelés fogalma

A részletes elmélet bemutatása előtt fontosnak tartom leírni, hogy mi a termelés fogalma. Termelés alatt azt a folyamatot értjük, melynek során egy termelőrendszerben előre definiált tervek alapján sokszorosan állítunk elő használati javakat. A sokszorosítást több környezeti feltétel is befolyásolja, ezek a következők: a tudomány, a technika és a technológia mindenkori szintje, valamint a társadalom, a gazdaság illetve a politika helyzete. A termelés magába foglalja a fizikai termékek előállításán kívül azokat a szolgáltatásokat is, amik elősegítik a termékek elkészülését, ilyen például a logisztika, a műszaki előkészítés és a minőségbiztosítás. [1]

2.1.2 A munka fogalma

‘’A munka az elvégzendő operációk vagyis műveletek halmaza. A termelésütemezési feladatokat tekintve a munka adott számú munkadarab együttesét jelenti.Egy megadott munkadarabok halmazán előre definiált műveleteket kell elvégezni, amit egy technológiai tervben adnak meg. A munkákhoz tartozó jelölés: J_i (i = 1,2,….,n). A J szimbólum reprezentálja a munkát (job). Az alsó indexben szereplő i a futóindexet, n a munkák számát jelölő szimbólum.”[2]

2.1.3 Az erőforrás fogalma

‘’Az ütemezési feladatokban az erőforrásokon olyan embereket, tárgyakat például gépeket értünk melyek alkalmasak a munkadarabokon való műveletek elvégzésére, ezáltal a munkákhoz tartozó operációkat elvégzik. Az erőforrás jelölése: M_r (r = 1,2,…….m). Az erőforrást az M szimbólum jelenti, rendszerint a machine angol szó rövidítése. Az alsó indexben szereplő r a futóindex, m az erőforrások számát jelölő szimbólum.”[2]

‘’Ütemezési feladatok jellemzői

Munka: Az előző alpontban megemlítettem.

Operáció (angolul operation): Ezzel az egységgel jellemezzük az ütemezést. Szokásos jelölése: Oi,j (i=1,2,…,n; j=1,2,n_i).

Műveleti idő: Időegységekben fejezi ki az operációk elvégzésének időtartamát. . Szokásos jelölése: pi,j (i=1,2,…,n; j=1,2, ni) (pi,j,k (i=1,2,…,n; j=1,2,..., ni ; $Mk \in \mu i,j$) - Ha a műveleti idő függ a választott erőforrástól)”[3]

‘’Erőforrás (angolul Resource): valós vagy virtuális "gép" (machine) entitás (gépek, dolgozók, munkahelyek stb.) amely képes elvégezni a szükséges műveleteket a munkadarabokon. jelölése: M_r (r=1,2,…,m).”[2]

‘’Gyártásütemezési feladatok megoldásának lépései

A gyártási folyamatok vizsgálata, analízise, az ütemezési feladatok modellezése, hatékony megoldási módszerek kidolgozása, végül alkalmazás (alkalmazásfejlesztés és integráció).

Az ütemezési modelleknek három alaptípusa van a prediktív a reaktív és a proaktív ütemezés.”[2]

‘’Ütemezési feladatok osztályozása

A szakirodalomban többféle osztályozási rendszer található. A legegyszerűbb a két jellemző tulajdonság csoportjával leírt feladatok osztályozása. A leggyakrabban alkalmazott szemlélet a háromelemes osztályozás, amely az erőforrásokat és azok bejárását elválasztja a munkák jellemzőitől és az egyéb végrehajtási korlátozások leírásától. Ennek a megközelítésnek a harmadik szempontja a célfüggvény, amely szerint összehasonlíthatók a megoldások és kereshető a legjobb megoldás. További megközelítések ennél finomabb felosztást is használnak. Léteznek négy vagy több elemre alapozott formális osztályozások is.”[2]

‘’Az ütemezési feladatok osztályozásának szempontrendszere: α|β|γ

Egy ütemezési feladat során a feladat rövid formális leírásához három alapvető kérdéskört kell megválaszolni. Ezen kérdésköröket a görög betűvel jelölt mezők jelzik.

> Az α - többértékű mező (szimbólumlista) jelenti az erőforrás-környezetet (machine environment), amely megadja az ütemezési feladatban szereplő erőforrások (gépek) jellemző tulajdonságait és a közöttük lévő kapcsolatrendszert, különös tekintettel az operációk végrehajtásának jellemzőire.
>
> β - többértékű mező (szimbólumlista) jelenti a munkák jellemzőit (job characteristics), amely megadja a munkákra vonatkozó korlátozásokat és végrehajtási jellemzőket.
>
> γ - a célfüggvényt vagy célfüggvényeket kijelölő szimbólumlista.
>
> Ezen mezők használatával kialakíthatjuk az ütemezési feladatok osztályozásának szempontrendszerét: α|β|γ.”[2]

‘’Az alábbiakban szeretném egy kicsit kifejteni ezen mezők jellemzőit, kezdve az α mezővel, ami az erőforrás környezetet jellemzi. α = α1α2 kételemű szimbólumlista. α1 ∈ {◦ , P, Q, R, F, FF, J, FJ, O, X, G, MPM...} az erőforrás-környezet szimbóluma . A ◦ szimbólum üresen hagyott mezőt jelent, ez egygépes esetben használatos. A szimbólumok jelentésének kifejtésére később kerül sor. α2 jelöli az erőforrások számát (◦, 1, 2, ..., k). Az egész szám konkrét darabszámú gépet jelent. A k szimbólum tetszőleges de választás után rögzített számú gépet jelent. A ◦ szimbólum üresen hagyott mezőt jelent, ilyen esetben a gépek száma tetszőleges lehet. Itt szeretnék egy pár környezeti fajtát bemutatni például: egyoperációs erőforrás-környezetek:

> egygépes modell,
>
> párhuzamosan működő gépek modellje.

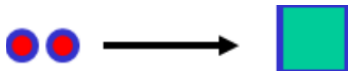
Többoperációs erőforrás-környezetek:

> Egyutas modell (Flow Shop),
>
> Rugalmas egyutas (Flexible Flow Shop),
>
> Többutas (Job Shop),
>
> Rugalmas többutas (Flexible Job Shop),
>
> Nyitott műhely modellje (Open Shop),

Vegyes műhely modell (Mixed Shop),

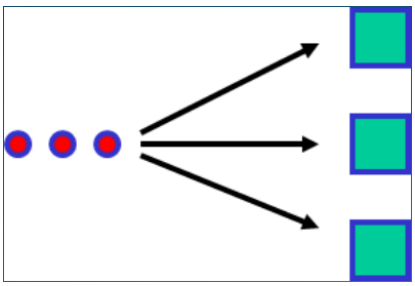
Általános műhely modell (General Shop).”[2]

‘’Az egyoperációs erőforrás-környezetek típusait szeretném egy kicsit jobban bemutatni. Az egyoperációs erőforrás-környezet esetében minden munkán egyetlen operációt kell elvégezni. Ezek közül lehet egygépes modell: Egy dedikált erőforrás végzi el minden munkán az operációt. Jelölése: $\alpha = 1$. Illusztráció az első ábrán. “[2]



1. ábra [2]

‘’A párhuzamosan működő gépek modellje: Több gép dolgozhat párhuzamosan különböző munkákon. Illusztráció a második ábrán.”[2]



2. ábra [2]

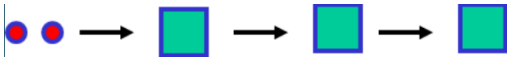
‘’Párhuzamosan működő gépek modellje esetén a gépek képességeitől függő változások az alábbiak lehetnek:

Teljesen egyenértékű gépek: A J_i munka műveleti ideje az M_r gépen csak a munkától függ ($p_{i,r} = p_i$).Jelölése: $\alpha 1 = P$,

Eltérő sebességű gépek: A J_i munka műveleti ideje az M_r gépen $p_{i,r} = p_i / s_r$ alakban írható fel, ahol a p_i csak a munkától függ, s_r az M_r gép sebessége. Jelölése: $\alpha 1 = Q$,

Független gépek: A J_i munka műveleti ideje az M_r gépen $p_{i,r} = p_i / s_{i,r}$ alakban írható fel, ahol a p_i csak a munkától függ, $s_{i,r}$ az M_r gép sebessége a J_i munka esetében. Jelölése: $\alpha 1 = R$

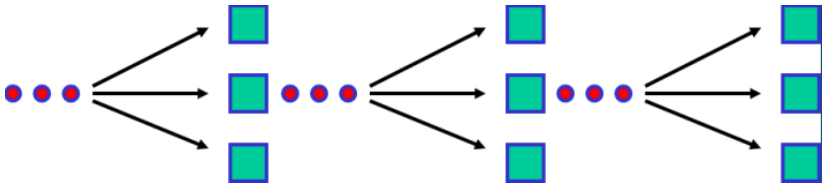
Illusztráció a harmadik ábrán.”[2]



3. ábra [2]

‘’Egyutas modell esetén a munkák operációinak száma és végrehajtási sorrendje minden munka esetében azonos. Az operációkat rendre egy-egy dedikált gépen lehet elvégezni. Jelölése: α1 = F

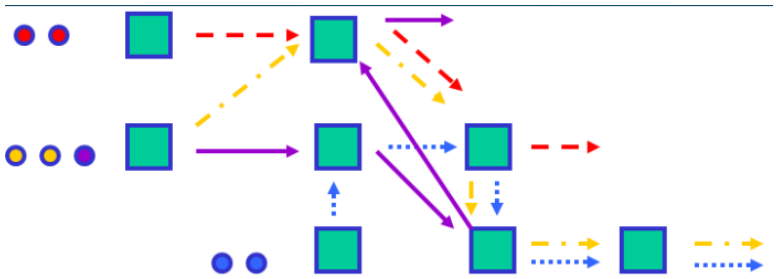
Rugalmas egyutas modell esetén a munkák operációinak száma és végrehajtási sorrendje minden munka esetében azonos. Az operációkat rendre egy-egy párhuzamosan működő gépcsoport bármelyik gépén el lehet végezni. Jelölése: α1 = FF. Illusztráció a negyedik ábrán.”[2]



4. ábra [2]

‘’Többutas modell esetén minden J_i (i=1,2, …, n) munka egy vagy több operációból állhat. Az operációk n_i darabszáma és végrehajtási sorrendje munkánként eltérő lehet. Adott J_i munka operációinak végrehajtási sorrendje kötött: $O_{i,1} \rightarrow O_{i,2} \rightarrow ... \rightarrow O_{i,ni}$ Több gép állhat rendelkezésre. Minden egyes operációt egy dedikált gép végezhet el. Jelölése: α1 = J

Rugalmas többutas modell esetén a Job Shop modellhez képest az eltérés abban nyilvánul meg, hogy az operációkat rendre egy-egy párhuzamosan működő gépcsoport bármelyik gépén el lehet végezni. Jelölése: $\alpha 1 = F_J$. Illusztráció az ötödik ábrán.”[2]



5. ábra [2]

‘’Nyitott műhely modell esetén hasonló a többutas modellhez (Job Shop), azzal a különbséggel, hogy a munkák operációinak végrehajtási sorrendjére nincs előírt korlátozás. Minden egyes J_i (i=1, 2, ..., n) munka operációi $O_{i,j}$ (j=1, 2, ..., ni) tetszőleges sorrendben elvégezhetők. Az ilyen feladatok megoldása során az ütemezéskor kell meghatározni az operációk sorrendjét is. Jelölése: α1 = O

Vegyes műveleti modell esetén a munkák egy részére a J (Job Shop) más részére az O (Open Shop) modell előírásai érvényesek. Jelölése: α1 = X”[2]

‘’Általánosított műhely modell esetén minden egyes J_i (i=1, 2, ..., n) munka több operációból állhat: $O_{i,j}$ (j=1, 2, ..., n_i). Az n_i operációk száma munkánként eltérő lehet. Több gép állhat rendelkezésre. Minden egyes operációt egy adott gép végezhet el. Jelölése: α1 = G. A G modell speciális esetei az X, O, J, F modellek.

Többcélú gép esetén az ütemezési feladatok szempontjából egy adott gép egynél több operáció elvégzésére is alkalmas. A rugalmas gyártórendszerekben gyakran egy adott gép több különböző szerszámot, készüléket, programot stb. használhat, így többféle operáció elvégzésére is alkalmas különböző időkben (átlapolódás nélkül). Az ilyen gépet az ütemezési feladatban többcélú gépnek nevezzük. Jelölése: α1 = MPM Példa: Legyen $\mu_{i,j} \subset \{M_1 , M_2 , \ldots, M_m\}$ azoknak a gépeknek a halmaza, amelyek alkalmasak a J_i munka $O_{i,j}$ operációjának végrehajtására. Egy $M_x \in \{M_1 , M_2 , \ldots, M_m\}$ gép MPM, ha van legalább két különböző $\mu_{i,j}$ és $\mu_{u,v}$ ($i \neq u$ vagy $j \neq u$) melyekre teljesül az, hogy $M_x \in \mu_{i,j} \cap \mu_{u,v}$.”[2]

‘’A β mező részletes kifejtése

A β mező a munkák végrehajtási jellemzőit és korlátozásait írja le. β = β1, β2, β3, β4, β5, β6, a szimbólumlista vesszővel elválasztott szimbólumokból áll. Minden szimbólumnak van saját jelentése. A szimbólumok opcionálisan szerepelhetnek a listában. Amelyik szimbólum szerepel a listában annak jelentése vonatkozik a feladatra. A lista lehet üres is, ilyenkor nincs definiált jellemző vagy korlátozás. Az alapértelmezett jellemzők mindaddig érvényben vannak, amíg egy definiált szimbólum jelentésével meg nem változtatja. Ilyen alapértelmezett jellemzők például a következők: egy gép egyszerre csak egy operációt végezhet, egy operációt egyszerre csak egy gép végezhet, az operációk nem szakíthatók meg, a munka az ütemezési időszak elejétől rendelkezésre áll, a munkák végrehajtása egymástól független stb. Néhány gyakran alkalmazott β szimbólum és jelentése:

pmtn (preemption) a munka végrehajtása megszakítható, majd később folytatható.

perm (permutation) a munkák sorrendje a gépek között nem változhat meg. Példa: F modell esetében előzés nélküli végrehajtást jelent, vagyis minden gépen a munkák végrehajtási sorrendje ugyanaz.

r_i (release date) a munkák (első operációjának) legkorábbi indítására külön előírás vonatkozik, az adott időpontnál korábban akkor sem kezdhető el a munka végrehajtása, ha egyébként minden más feltétel teljesül.

d_i (due date) a munkák (utolsó operációjának) befejezésére külön előírás vonatkozik. Ennek speciális esete az, amikor a munkákra egy közös határidő vonatkozik, ilyenkor d szimbólum szerepel a listában. ”[2]

‘’$p_i = 1$ ($p_{i,j} = 1$) (processing time) a munkák operációinak műveleti idejére vonatkozó előírás: egységnyi műveleti idő alatt teljesíthető minden operáció.

s_i ($s_{i,l}$) ($s_{i,l,k}$) (setup time) az operációk megkezdése előtt a gépeket megfelelően be kell állítani, a beállításra fordított időtartam nem hanyagolható el. Példák: s_i az átállítás ideje csak a soron következőp J_i munkától függ. $s_{i,k}$ az átállítás ideje az aktuális M_k géptől és a következő J_i munkától függ, $s_{i,j,k}$ az átállítás ideje az aktuális M_k géptől és az utolsó befejezett J_i munkától valamint a következő J_j munkától függ.”[2]

‘’A_i (machine assignments) a munkák gépekhez rendelésére vonatkozó korlátozás, amely előírja a műveletvégzésre alkalmas gépek halmazát (pl.: párhuzamos gépek esetében). prec (precedence) a munkák végrehajtásának sorrendjére szigorú előírás vonatkozik, amely általános alakban egy irányított, körútmentes gráffal jellemezhető G=(V, A). A gráf csúcspontjai jelentik a munkákat V={1,2, …, n}, az irányított élek a munkák között fennálló kötelező sorrendiséget adják meg: (i, l) ∈ A. Az i → l jelentése: az J_i munka utolsó operációjának befejezése után kezdődhet a J_l munka első operációja. Ennek az általános esetnek további speciális eseteit szokás megkülönböztetni és önálló szimbólummal jelölni: intree A G gráf egy gyökeres fa, amelynek minden csúcspontjának kifoka (a belőle induló élek száma) legfeljebb egy. outtree A G gráf egy gyökeres fa, amelynek minden csúcspontjának befoka (az oda vezető élek száma) legfeljebb egy. tree A G gráf vagy egy intree vagy egy outree. chains A G gráf élek láncolatának halmaza, minden csúcsnak a befoka és a kifoka is legfeljebb egy. sp-graph A G gráf egy soros-párhuzamos gráf, amely részgráfokból rakható össze párhuzamos vagy soros kapcsolással. Párhuzamos kapcsolással: $G = (V_1 \cup V_2 , A_1 \cup A_2)$. Soros kapcsolás: $G = (V_1 \cup V_2 , A_1 \cup A_2 \cup T_1 \times S_2)$, $T_1 \subseteq V_1$, $S_2 \subseteq V_2$. Néhány formális: leírás az α és β szimbólum ismeretében:

$1||\sum C_i$ $1||\sum(w_i C_i)$ $1|d_i|T_{max}$ vagy $1|d_i|Lmax$ $1|d_i|U_i$ 1| prec, d_i | L_{max} .

Egyoperációs erőforrás-környezetek: Párhuzamos gépes esetek

$P \,||\, \sum_i C_i$ P || Cmax P | d_i | L_{max} P | p_i=1;

r_i=egész | L_{max} P(s) | p_i=1; r_i=egész;

d_i =egész | L_{max}.

Többoperációs erőforrás-környezetek:

Flowshop esetek

$F||C_{max}$

$F|perm|C_{max}$

$F2|perm|C_{max}$

$F3|perm|C_{max}$

$F_m|perm|C_{max}$ m>3 (Palmer, Dannenbring, CDS)”[2]

‘’A γ mező részletes kifejtése

A γ mező a célfüggvényeket jelenti Egy ütemezési célfüggvény egy számítási eljárás, amely egy ütemterv adott szempont szerinti minőségét fejezi ki numerikus formában. A célfüggvények által eredményül adott értékek teszik lehetővé az ismert ütemtervek (megoldások) értékelését és összehasonlítását.

Definíciók és jelölések:

Alapadatok:

Munkák (jobs): J_i (i=1,…,n)

Határidők (due date): d_i

Ütemezéstől függő jellemzők:

Tényleges indítási időpontok (Start time): r_i jelöli a J_i munka első operációjának indítási időpontját.

Befejezési időpontok (Completion time): C_i jelöli a J_i munka utolsó operációjának befejezési időpontját.

Jellegzetes ütemezési célfüggvények:

Átfutási idő (Flow time): $F_i = C_i - R_i$

Késés (Lateness): $L_i = C_i - d_i$

Csúszás (Tardiness): $T_i = \max\{0, L_i\}$

Sietés (Earliness): $E_i = \max\{0, -L_i\}$

Egységnyi büntetés (Unit penalty): $U_i=\{1$ ha $C_i > d_i$ 0 egyébként $\}$."[4]

''Az előzőekben példaként feltüntetett jellemzők (C_i , F_i , L_i , T_i , E_i , U_i ,) bármelyikét G_i helyett beírva a következő képletekbe megkaphatjuk a munkákhoz kapcsolódó legfontosabb célfüggvényeket:

Legnagyobb (maximum): $\gamma=max_{i=1}^{n}\{G_i\}$

Összeg (total): $\gamma=\sum_{i=1}^{n} G_i$

Átlag (average): $\gamma=\frac{\sum_{i=1}^{n} G_i}{n}$ "[4]

''Abban az esetben, ha meg akarjuk különböztetni a munkákat fontosságuk alapján, prioritásértékeket (vagy súlyokat) rendelhetünk hozzájuk, és ezeket figyelembe vehetjük a célfüggvények számítása során.

Legnagyobb (maximum): $\gamma=max_{i=1}^{n}\{w_iG_i\}$

Összeg (total): $\gamma=\sum_{i=1}^{n} w_iG_i$

Átlag (average): $\gamma=\frac{\sum_{i=1}^{n} w_iG_i}{n}$

Néhány határidőhöz nem kapcsolódó gyakran használt célfüggvény:

Legkésőbbi befejezési időpont (Makespan): $C_{max} =max_{i=1}^{n}\{C_i\}$

Befejezési időpontok összege: $C_{sum}=\sum C_i = \sum_{i=1}^{n} C_i$

Végrehajtási idő (Execution time): $E_t=max_{i=1}^{n}\{C_i\}- min_{i=1}^{n}\{r_i\}$

Néhány gyakran használt határidőhöz kapcsolódó konkrét célfüggvény

A legnagyobb csúszás (the maximum tandiness): $T_{max} =max_{i=1}^{n}\{T_i\}$

A csúszások összege (total tardiness): $T_{sum}=\sum T_i = \sum_{i=1}^{n} T_i$

A késő munkák száma (the number of tardy jobs): $U_{sum}=\sum_{i=1}^{n} U_i$

A súlyozott termelési pontosság (weighted production timeliness): $P_t=\sum_{i=1}^{n} u_i * E_i + \sum_{i=1}^{n} v_i * T_i$

Az u_i és a v_i nemnegatív valós számok, melyek a J_i munka sietésének és csúszásának súlyait jelölik.”[2]

2.1.4 Az ütemezés fogalma

‘’A rendelkezésre álló erőforrások között kell elosztanunk az elvégzendő munkákat. Az erőforrások típusai és a munkák eltérhetnek egymástól és erősen függnek a vizsgálandó szituációtól. “[2]

2.1.5 A gantt diagram definíciója

‘’A Gantt diagram egy idődiagram, melyet arra használunk, hogy ütemezési feladatokat jellemezzünk. Két tengelye van egy vertikális és egy horizontális. A vertikális jelzi az aktivitásokat (műveleteket). Az aktivitásokat téglalapokkal jellemezzük. A horizontális tengely az eltelt időt jelzi. A téglalapok kezdete jelzi az aktivitások kezdő időpontját. A téglalapok vége pedig az aktivitások végét. A téglalapok szélessége jellemzi az aktivitások időtartamát. A programomban viszont nem aktivitásokkal, hanem elvégzendő munkákkal dolgozom. A felhasználói felületen megjelenő munka kifejezés a project aktivitását (műveletét) jelenti. A Gantt diagram sémája a munkák ugyanolyan módon való leírására is alkalmas. Amennyiben gyártásütemezési feladatot képezünk le projektütemezési feladatra, akkor az eredeti gyártási munka operációi jelentik a projektütemezési feladatban a munkákat (aktivitásokat).”[2]

2.1.6 A projekt definíciója

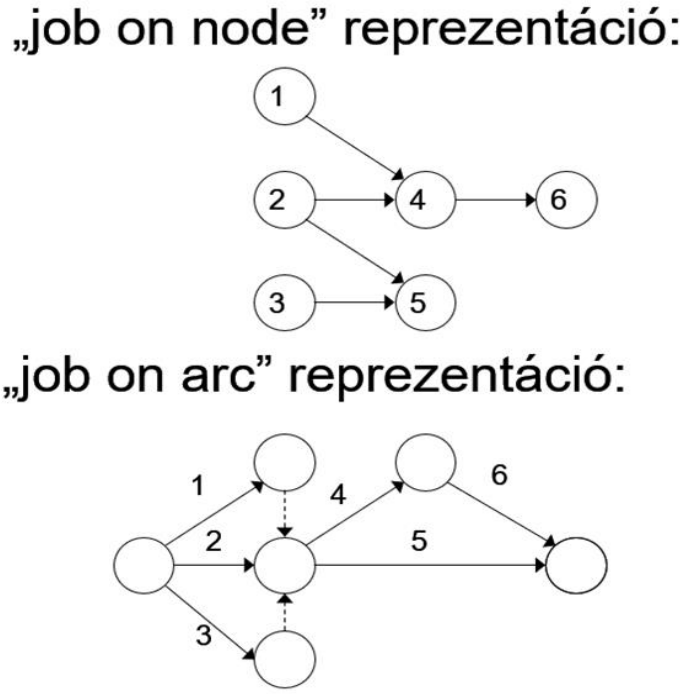
‘’Termék elkészítésére vagy szolgáltatás nyújtására irányuló törekvés, amely általában nagyszámú komponens feladat/aktivitás végrehajtását igényli.”[2]

A projektütemezés fogalma

‘’Úgy tervezzük meg a projektek időbeli végrehajtását, hogy az előírt korlátozásokat figyelembe véve teljesítsük a megfogalmazott célokat. A projektütemezés célja az egy vagy több célú optimalizálás, melyben sokféle szempont szerepelhet. A projektütemezés során feladatok és aktivitások hálózata alakul ki, például megelőzési relációk alapján, minden esetben figyelembe kell venni a korlátolt és korlátlan erőforrásokat. A modellek és megoldási

módszerekhez tartozik még továbbá a kiterjesztett modellek és módszerek, amelyek az összetett kategóriába tartoznak. A következő képen a projekt egyik ábrázolási módja látható. ”[3] A hatodik ábrán.

Feladat	p(j)	Előfeltétel
1	2	-
2	3	-
3	1	-
4	4	1, 2
5	2	2, 3
6	1	4



6. ábra [3]

’’Erőforráskorlátozások nélküli projektütemezési feladatok esetében azt feltételezzük, hogy párhuzamosan és korlátlanul állnak rendelkezésre az erőforrások, a megadott n számú feladat megelőzési relációkat tartalmaz, a feladatok p_j végrehajtási idejét ismerjük minden esetben. Az ütemezés célja minimalizálni a projektek befejezési időpontját. Ebben az esetben a j feladat végrehajtási ideje: p_j legkorábbi lehetséges kezdési időpontja: S_j' , legkorábbi lehetséges befejezési időpontja: C_j' , legkésőbbi megengedett befejezési időpontja: C_j'' , időtartaléka: $slack_j = C_j'' - p_j - S_j'$

A kritikus feladatnak nincs időtartaléka $slack_j = 0$”[3]

’’A kritikus útvonal a kritikus feladatok láncolata

A kritikus útvonal módszer angol rövidítése a CPM, ezen módszer két algoritmusból, áll az előre haladó eljárásból és a hátrafelé haladó eljárásból. Az előre haladó eljárás kiindulási pontja a precedencia gráf kezdeti pontja, majd a gráf további elemein végighaladva az irányított élek mentén kiszámítja minden feladat esetében a legkorábbi megengedett indítási és befejezési időpontot, majd az utolsónak elkészülő feladat adja meg a projekt befejezési időpontját. Az eljárás lépései: Legyen $t = t_s$ A megelőző feladattal nem rendelkező minden egyes j feladat

esetében legyen $S_j' = t$ és $C_j' = t + p_j$. A megelőző feladattal rendelkező minden egyes j feladat esetében legyen induktív módon: és $C_j' = S_j' + p_j$. A legkorábbi projekt-befejezési időpont: $C_{max} = \max\{C_1', C_2', \ldots, C_n'\}$

A visszafelé haladó eljárás kiinduló pontja a projekt befejezési időpontja és a menete során a precedencia gráfon az irányított élek mentén visszafelé haladva kiszámítja minden feladat esetében a legkésőbbi megengedett befejezési és indítási időpontot tekintettel arra, hogy a projektbefejezési határidő még tartható legyen. A visszafelé haladó eljárás lépései a következők: Legyen $t = C_{max}$ A rákövetkező feladattal nem rendelkező minden egyes j feladat esetében legyen $C_j'' = C_{max}$ és $S_j'' = C_{max} - p_j$. A rákövetkező feladattal rendelkező minden egyes j feladat esetében legyen és $S_j'' = C_j'' - p_j$. Ellenőrizzük, hogy $t_s = \min\{S_1'', \ldots, S_n''\}$.

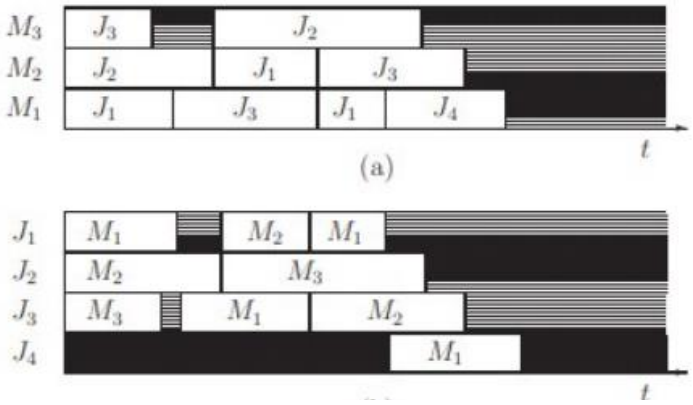
A kritikus útvonal módszerben az előrehaladó eljárás megadja az S_j' megengedett legkorábbi indítási időpontját minden feladatnak. A hátrafelé haladó módszer megadja az S_j'' megengedett legkésőbbi indítási időpontját minden feladatnak. Ha ezek azonosak akkor a feladat kritikus. Ha ezek különbözőek, akkor a feladatnak van tartaléka (**slack)**. Kritikus útvonal (**critical path):**
kritikus feladatok láncolata, amely a t_s kezdési időponttól a C_{max} befejezési időpontig vezet. Kritikus útvonalból egyszerre több is lehet, ezek akár részben fedhetik is egymást.”[3]

‘’A projektütemezési módszerekkel megoldhatunk munka ütemezési problémákat is, viszont akkor a kijelölt feladatokra kell projektként tekintenünk, és a rendelkezésre álló gépekre, mint a projektütemezési feladatok erőforrásaira. Például a korábban bemutatott Flow Shop, Job Shop, Open Shop, Mixed Shop és General Shop ütemezési problémák leképezhetők projektütemezési feladatra. Minden munka együtt egy nagy projektet alkot, amelyben az eredeti munkák operációi lesznek a project aktivitásai. Az operációk végrehajtására kijelölt dedikált gépek lesznek a projektütemezési feladat erőforrástípusai, melyekből rendre egy egységet igényel az aktivitás végrehajtása.”[3]

2.2 Termelésütemezés

Tegyük fel, hogy m darab gépnek el kell végezni n darab munkát (Job-ot). Az ütemterv az egyes munkák számára egy vagy több időintervallum (előkészítési és operációs idő) kiosztását jelenti, egy vagy több gépre. Fontos, hogy egy adott Job-hoz több operáció is tartozhat. A kész terveket végül Gantt-diagrammon ábrázoljuk. Ennek két lehetséges verziója

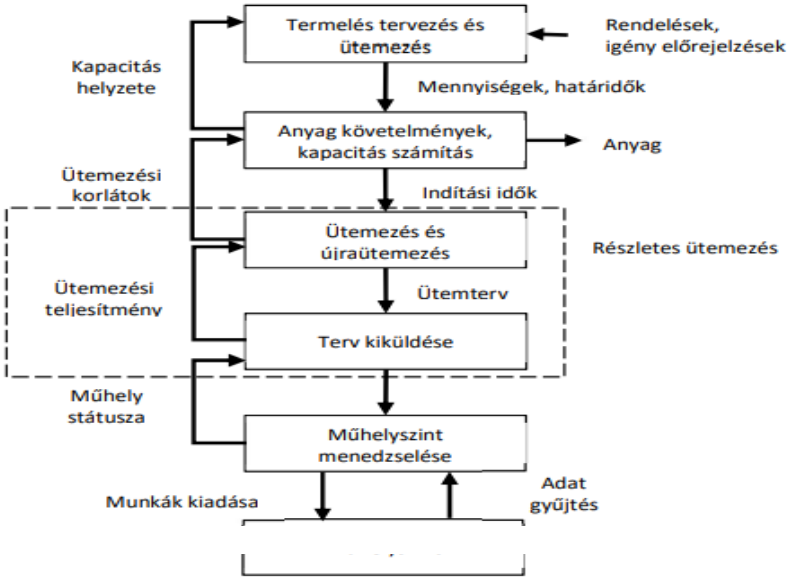
is van, az egyik a gép-orientált, a másik pedig a munka-orientált megjelenítés. A hetedik ábrán jól látszik a kettő közötti különbség, az „a” részen látható a gép-orientált nézet, ahol a gépekhez rendeljük hozzá a munkákat, miközben a „b” részen a munkákhoz rendeljük a gépeket. Mindkét esetben a diagram mérete az szükséges idő nagyságát jelzi [4].



2. ábra: Gantt diagram alaptípusai: „a” gép-orientált, „b” munkáka-orientált nézet. [10].

7. ábra: Gantt diagram alaptípusai: „a” géporientált, „b” munkaorientált nézet [4]

‘’Vegyünk egy általános gyártókörnyezetet, amelyben szerepelnek a megrendelések és a határidők is. Megfelelő információkkal dolgozva, az ütemezés folyamata a nyolcadik ábrán látható.”[4]



3. ábra: Termelés tervezés és ütemezés folyamatábrája [14]

8. ábra: Termelés tervezés és ütemezés folyamatábrája [5]

2.3 Projektütemezés

”A gyakorlatban vannak személyzeti és munkaerő korlátozások. Az erőforrástípusok számos különböző egységekből állnak és minden egység megadott operációt tud elvégezni, arra specializálódott tudással. Minden munka elvégzéséhez egy megadott számú egység szükséges. Ha a feldolgozandó munkák közül néhány egymásra esik időben, akkor az erőforrás igényük összege meglehet, hogy nagyobb lesz, mint amit az elérhető egység képes feldolgozni. A cél az lenne, hogy az ilyen eseteket minimalizáljuk. Ezen problémát követi a projekt ütemezés munkaerő korlátozásokkal. Ennek a problémának a megfogalmazásához további jelölések szükségesek. Legyen N az egységek száma a munkaerőben. Legyen W_i az egységekre osztott i. egység száma, amire kiosztottuk az operációkat, és W_{ij} az i - dik egység j – dik operációja.

Egy példa: Nézzük a következő példát, ahol öt munka és két féle operáció van. Négy egyes kategóriájú és nyolc kettes kategóriájú. A feldolgozási idő és a munkamenet szükségleteit az alábbi táblázat mutatja: kilencedik ábra

Jobs	1	2	3	4	5
p_j	8	4	6	4	4
W_{1j}	2	1	3	1	2
W_{2j}	3	0	4	0	3

9. ábra [4]

A precedencia korlátozásokat pedig az alábbi táblázatba helyeztem el: tizedik ábra

Job	*Immediate Predecessors*	*Immediate Successors*
1	—	4
2	—	5
3	—	5
4	1	—
5	2,3	—

10. ábra [4]

Ha nem lennének munkaerő korlátozások akkor a kritikus út az $1 \Rightarrow 4$ és a $C_{max} = 12$. Viszont nincs elfogadható megoldás ami kielégíti a $C_{max} = 12$ célfüggvényt a munkaerő korlátozásokkal. Az optimális ütemezés a $C_{max} = 18$. Erre az ütemezésre alapozva a kettes és a hármas a [0,6] intervallumon kerülnek feldolgozásra, az egyes és az ötös munka a [6,14]

intervallumon kerülnek feldolgozásra és a négyes munka pedig a [14,18] intervallumon kerül feldolgozásra. A szimpla projektütemezési problémákkal szemben a munkaerő korlátozott projektütemezési problémák jelentősen bonyolultabbak.

$$W_{ij} = \sum_{u=t}^{t+p_j-1} \mathrm{x_{ju}}$$

Erre a problémára nem létezik lineárisan programozható megoldás, ezzel ellentétben létezik egy egész számokat használó formula a feladat megoldásához. Ahhoz, hogy ezt a problémát egész számos módsszerel fel tudjuk írni, azt feltételezzük, hogy minden feldolgozási időt fix megadott érték és egész szám reprezentál. Egy kezdetleges elemnek n + 1 egy nulla feldolgozási idővel. Az n + 1 munka megelőzi az összes többi munkát, például amelyhez nem tartozik olyan rákövetkező elem, ami az n + 1 munka felé tartana. Legyen x_{jt} egy 0 vagy 1 érték, ahol az 1 – es jelzi, ha a j munka befejeződött egy t időpontban és 0 ellenkező esetben. Vagyis a j munkákhoz tartozó operációk az i egységről a [t-1,t] intervallumban:

Legyen H a projekt teljes időtartamának felső határa. Egy egyszerű, de nem túl szoros határ kiválasztása megadható a következő képlettel:

$$\sum_{t=1}^{H} tx_{jt}$$

Tehát a j munka befejezési idejét a következő képlettel kifejezhetjük:

$$H = \sum_{j=0}^{n} \mathrm{p_j}$$

A projekt időtartamát is meghatározhatjuk:

$$\sum_{t=1}^{H} tx_{n+1,t},$$

A számolási formula tehát kifejezhető az alábbiakkal:

Minimalizáljuk a

$$\sum_{t=1}^{H} tx_{n+1,t},$$

Elemet.

$$\sum_{t=1}^{H} t\, x_{jt} + p_k - \sum_{t=1}^{H} \mathrm{tx_{kt}} \leq 0$$

minden j → k ∈ A (j-ből k-ba, k eleme A)

$$\sum_{j=1}^{n} \left(\mathrm{W_{ij}} \sum_{u=t}^{t+p_j-1} x_{ju} \right) \leq W_i$$

minden i és t-re

$$\sum_{t=1}^{H} \mathrm{x_{jt}} = 1$$

minden i -re

Ennek a módszernek a célja az, hogy a projekt időtartamát minimalizáljuk. Az első kiosztott korlátozások biztosítják hogy a precedenciára vonatkozó korlátozások érvényesek, például ha egy j munka egy k munka után következik, akkor a k munka végrehajtási ideje nagyobb vagy egyenlő kell, hogy legyen a j plusz p_k munkáénál. A második kiosztott korlátozások biztosítják, hogy az i egységből való összes követelmény egy t időpontban nem haladja meg az akkor elérhető egységek számát. A harmadik kiosztott korlátozások azt biztosítják, hogy minden munka feldolgozásra kerül. Mivel ez a megoldás nagy mennyiségű munkák és hosszú időhorizont esetén, elég nehezen megoldható. Viszont egy pár fontos speciális esetre heurisztikus módszereket fejlesztettek ki, amik elég hatékonynak bizonyultak az elmúlt időkben.”[4]

2.4 Az RCPSP probléma és alapvető megoldási módszerei

”Az erőforrásokban korlátozott projekt ütemezési problémája (angolul Resource-Constrained Project Scheduling Problem) a következő jelöléseket használja:

n a munkák száma j=1,...,n

N az erőforrások száma i=1,...,N

R_k a k erőforrás elérhetősége

P_j a j munka időtartama

R_{kj} a k erőforrás, ami a j munkához szükséges

P_j a j munka elődje, vagyis azon munka, ami a j munkát előzi meg végrehajtási sorrendben

Az RCPSP célja, hogy az időtartamot minimalizálja: $C_{max} = \max C'_j$

Korlátozásai:

egyik munka sem kezdődhet T = 0 előtt

tartania kell a precedencia által meghatározott kapcsolatokat

véges erőforrás kapacitással dolgozik

Elválasztó ívek: Olyan munkapárosítások melyek nem végezhetőek ugyanabban az időpontban.

A prioritásos szabály alapú ütemezés

- Generált séma
  - soros
  - párhuzamos
- Prioritás szabály
  - Legutolsó befejezési idő
  - Minimum időtartalék

Soros ütemezési metódus

Minden lépés egy munkát reprezentál ⇒ n lépés

- elkészült munkák halmaza: ütemezett munkák
- döntési halma, azon munkák ahol a megelőző munkák már ütemezésre kerültek
- megmaradt halmaz: minden más munka

Eljárás:

1. Egy üres ütemezéssel való kezdés
2. Kiválasztjuk a legmagasabb döntési prioritással rendelkező munkát és beütemezzük, amilyen hamar csak lehetséges
3. Ha nem üres a döntési halmaz, akkor a második lépéssel folytatjuk

Párhuzamos ütemezési metódus

- Maximum n lépés
- Minden n lépés reprezentál:
    1. részleges ütemezést
    2. t_n ütemezési időt
    3. négy egymástól megkülönböztetett munkahalmaz:
        - beütemezett munkák halmaza: beütemezett munkák t_n időpontban elkészített
        - aktív halmaz: beütemezett munkák, amelyek még nem készültek el
        - döntési halmaz: minden be nem ütemezett munka, ami lehetséges, hogy már ütemezésre került
        - megmaradt halmaz: minden be nem ütemezett munka, amit nem lehet ütemezni

Eljárás:

1. Egy üres ütemezéssel való kezdés
2. Legyen T az első idő ahol egy még be nem ütemezett munka elkezdődhet. Legyen D azon munkák halmaza, amelyek elkezdhetők T időpontban, és minden előfeltétele már teljesült.
3. Válasszuk ki a legmagasabb prioritású munkát a D halmazból és a T időpontra ütemezzük.
4. Ismételjük újra a 2. lépést, amíg vannak ütemezendő munkák

Prioritási szabályok:

Legutoljára elvégzett (LFT) prioritási szabály:

$$V_j = -C_j''$$

Minimum tartalékidő (MS) prioritási szabály:

Vj = - (C_j'' - p_j - S_j'*), ahol S_j'* az akkor leghamarabbi indítási idő''[6]

A bemutatott felépítő algoritmusok és prioritási szabályok az RCPSP probléma egyik alapvető megoldási koncepciójára mutatnak példákat. A szakirodalomban többféle megoldási módszert javasoltak:

'' Az első RCPSP optimalizálási modell a [7]-ben volt közzétéve. A [8] -ban fedezték fel, hogy az RCPSP nem polinomiális idő alatt megoldható probléma.A [9] -es forrásban több féle mérési eredmények és RCPSP kiterjesztések találhatóak. A főbb kiterjesztések a következőek:

(1) Az aktivitások általánosítása

(2) Alternatív precedencia korlátozások és hálózati karakterisztikák

(3) Több projekt figyelembevétele

A többfajta célfüggvény támogatásának megközelítése a [10] irodalomban található. Számos metaheurisztikus keresési algoritmust is használhatunk az ütemezési feladatok megoldására. Példaként a gyakran használt metaheurisztikus halmazok közé tartozik a részecske raj algoritmus[11], a tabu keresés [12], a genetikus algoritmus [13], a memetikus algoritmus [14], a differenciált evolúciós algoritmus [15], a Jaya algoritmus[16]"[17]

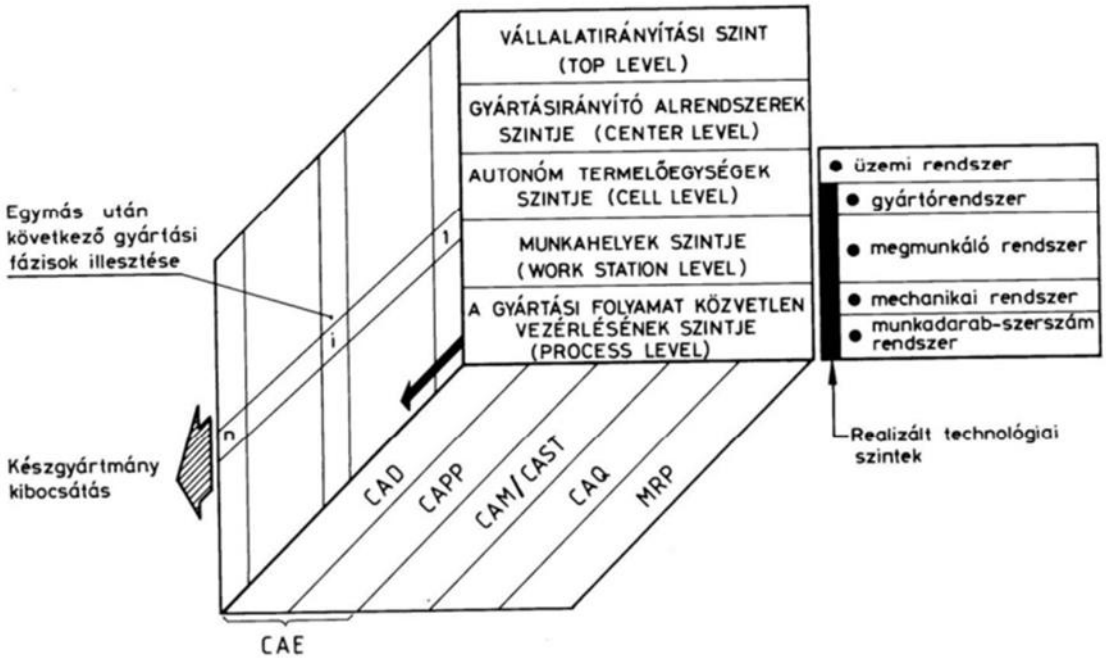
2.5 A termelésinformatika szerepe

A termelésinformatika a termelési rendszerek és folyamatok számítógépes modellezésével, tervezésével és irányításával foglalkozó alkalmazott informatikai tudományterület. [1] A termelési rendszerek hatékonyságának növelésére kiválóan alkalmazhatók az informatikai eszközök, modellek, módszerek és technológiák. A termelési folyamatokra viszonylag magas költségek jellemzőek, továbbá általában bonyolult és változatos összetételű termékeket állítanak elő különböző korlátozottan rendelkezésre álló erőforrások felhasználásával, melynek során rendkívül nagymennyiségű információt használnak fel a folyamatok tervezéséhez, irányításához és felügyeletéhez. [18] A termelés magában foglalja a fizikai termékek előállítását, a kapcsolódó szolgáltatásokat, a teljes műszaki előkészítést és a minőségbiztosítást is. A gyártás egy olyan állapot-transzformáció, amelyben a munkadarabok a legdurvább nyersállapotból a legfinomabb készállapotba kerülnek át. A gyártás ilyen értelemben a termelés részhalmazának tekinthető. [19]

2.6 Számítógéppel integrált gyártás

“A CIM (számítógéppel integrált gyártás) egy csúcstechnológiai megközelítés a hatékonyabb gyártáshoz, amely a digitális számítógépek sebességét és pontosságát használja fel integráló tényezőként a teljes gyártási folyamat minden fázisában. A legszélesebb értelemben véve, a CIM a piaci igények elsődleges felismerésétől és a termék koncepciójától kezdődően kiterjed a teljes gyártási folyamatra és a kereskedelmi szférában, a késztermékeknek a vevőhöz (megrendelőhöz) való kiszállításával (delivery) fejeződik be. A CIM lényegét az integrációban kell keresnünk, amely a következő elemek magasabb fokú szintézisét jelenti:

- Időbeli: Az egymást követő gyártási operációk úgy kövessék egymást, hogy a készgyártmány-kibocsátás optimális legyen.
- Szervezeti / architekturális: Az egymás feletti irányítási szintek integrációja.
- Funkcionális: Az egymás mellett működő vállalati funkciók integrációja. Vizuálisan a következő ábra szemlélteti a hármas integrációt.



11. ábra: Az alábbi ábra a hármas integrációt szemlélteti [3]

Az évek alatt több CIM modell látott napvilágot, ami vizuálisan reprezentálja az adott vállalat működését, ezzel felállítva egy alapot, amit más vállalatok integrálhatnak a saját termelőkörnyezetükbe.” [3]

2.7 Egy webes alkalmazás felépítése

A webalkalmazások általában két részből épülnek fel: egy kliens és egy szerver modulból. A szakirodalom ezeket frontend és backend angol szavakkal jellemzi. A szerver része felelős a felhasználó által kért adatok kiszolgáltatásáért. A kliens része pedig azért felelős, hogy a felhasználó el tudja küldeni a kéréseit valamilyen módon, amelyet a kliens előállított. A kliens többféle technológiával előállítható. Egyik gyakori módszer az, amely során három különböző nyelvvel formálható a kliens. Az egyik a HTML (Hyper Text Markup Language) ami azért felelős, hogy meghatározzuk a weboldal elemeit, a CSS (Cascading Style Sheet), ami azért felelős, hogy a HTML segítségével létrehozott elemek stílusait azaz színezésüket, méreteiket, pozícióikat megváltoztassuk, és végül a JavaScript, ami azért felelős, hogy a felületen való eseményekhez funkciókat rendeljen. A szerver rész formálásához szintén többféle technológia használható. Egyik gyakori módszer az, amely során tisztán egyetlen programnyelvet használunk, amelyik rendelkezik egy hozzá alkalmas csomaggal, vagy ha nem, akkor azon csomagot magunknak kell elkészítenünk.

Az én alkalmazásom kliensébe a JavaScript helyett a C# programnyelvet választottam. A szerver programnyelve pedig a C++.

3. A feladat modellezése

3.1 Kliens szerver koncepció

A kliens szerver koncepcióban a kliens és a szerver két különböző alkalmazás. Tekinthetjük őket rétegeknek is. A rétegek egy program esetében cserélhetőek. A kliens felel a felhasználói felület megjelenítéséért, a szerver pedig a felhasználó által megválasztott számítási feladatok elvégzéséért. A felhasználó csak a kliens felületével van közvetlen kapcsolatban, tehát csak azt látja, amit a kliens meghatározott. A felhasználónak viszont szüksége van a szerver által biztosított funkciók elérésére. A kliensnek tehát továbbítania kell a felhasználó

által igényelt kéréseket, ezen feladat megvalósításához szükséges a kliens és szerver között valamilyen fajta kommunikáció kialakítása.

3.2 Kommunikáció

A szerver és kliens közti kommunikációra a http hívásokat szokás használni. A http hívások általában JSON formátummal dolgoznak, vagyis a modulokat kezelő elemeknek fel kell tudniuk dolgoztatni a JSON formátumot. Ezen megvalósításhoz a szervernek egy hálózati portot kell megnyitnia a gazda számítógépen, a kliensnek pedig az adott programnyelvhez igazított kliens objektummal el kell érnie a megadott szervert.

3.3 Szerverrugalmasság biztosítása

A szerver rugalmasságának biztosításához a konténerizációhoz folyamodtam. A konténerizáción belül is a Docker programhoz. A Docker programról és használatáról a későbbiekben fogok beszámolni, előljáróban, csak annyit, hogy virtuális számítógépeket hozhatunk vele létre, és, ha több számítógép futtatja egyszerre ugyanazt a programot, akkor rugalmasabb elosztani rajtuk a terhelést, terheléselosztás szempontjából mindegy, hogy az adott számítógép fizikai vagy virtuális.

4. Szoftver tervezése

4.1 Főbb szoftver modulok

Dolgozatomban négy főbb modult szeretnék megemlíteni. Az első az ütemezési modul, amely a bejövő adatokból elkészíti az ütemezési tervet. A második a szerver modul, ami a beérkező hívások feldolgozására szolgál. A harmadik a kliens modul, ami a beérkező kérések elküldésére szolgál. A negyedik pedig a karbantartó modul, ami az alkalmazás futtatását biztosítja.

4.2 Ütemezés megoldó modul

Amint az előző pontban említettem ezen modul felel az ütemezési feladatok megoldásáért. A precedenciagráf, a célfüggvény kiértékelés, és a munkák sorrendje itt kerül kiszámításra illetve megtervezésre. Itt szeretném megjegyezni, hogy a kód ezen modulja nem az én érdemem, Kulcsár Gyula tanár Úr által elkészített programot használtam újra, ami eredetileg egy konzolos alkalmazás volt. A modul C++ programozási nyelvvel készült, a modulhoz tartozó programozási nyelven nem tudtam változtatni, hiszen az egész modul átírása túl sok ídő lett volna, és nem készültem volna el vele a kiírt határidőre. A programrész tehát C++ programnyelvben maradt. Az ütemezési algoritmusok programozására bármilyen magas szintű programnyelv alkalmas, de célszerű objektum orientált nyelveket választani, hiszen az objektum orientáltság által biztosított lehetőségek nagyban elősegítik a fejlesztést.

4.3 Szerver modul

A szerver modul fő feladata a beérkező hívások kezelése. A fő feladathoz hozzátartozik a paraméterek, adatok kinyerése a kérésből. Az adatokat egyetlen string típusú változóból nyerjük ki, a változó egy fájl szövege, a fájl formátumát és szövegformátumát az ütemezés megoldó modulhoz kellett igazítanom. Az adatok elküldésére több megoldás is lehetséges: mivel http kéréssel dolgozik a szerver ezért a legegyszerűbb megoldás a POST kérés, ahol a body részében adjuk meg JSON string változóként a fájl tartalmát, továbbá GET kérést is használhatnánk; de mivel a GET kérés paramétereit az url-ben kell elhelyezni, az egy nagyon hosszú string változót eredményezne. A szerver modulnak fel kell tudnia dolgozni a beérkező paramétereket, majd továbbitania kell feldolgozható formában az ütemezés megoldó modulnak. Tudnia kell még előállítania a választ a kiválasztott formátumban, fel kell készülnie az esetleges hibakezelésre is, ha a bemeneti adatok nem lennének helyesek formailag vagy tartalmilag. Minden http protokollal kompatibilisnek kell lennie. A http fejlécek előállítására is képesnek kell lennie.

4.4 Kliens modul

A kliens modul feladata, hogy kéréseket küldjön a szervernek. Ezen modul eleme a felhasználói felület. A felhasználói felület elemein keresztül küldhetünk kéréseket a szervernek, ezen elemek nagyrészt gombok. A kliensnek továbbá tartalmaznia kell egy kliens objektumot,

mely a szerverrel való kapcsolat létesítésére szolgál. A kérések elküldése ezen objektum metódusai egyben. A kliens eléréshez egy böngésző vagy a curl nevü program szükséges. A kliens az egyetlen egy felület amivel a felhasználó programozói tudás nélkül hozzáférhet. A kliens használatbavehető egyetlen egy linken.

4.5 A konténerizált alkalmazás karbantartó modulja

A karbantartó modul feladata a konténerizált alkalmazás futtatása. Felkészülnek kell lennie az alkalmazás újraindítására túlterhelés esetén. A virtuális gép előállítására, futtatására, leállítására is alkalmas kell, hogy legyen. A virtuális gép saját hálózattal rendelkezik,valmint belső tűzfallal is.

4.6 A szoftvertervezéshez szükséges osztályok

Az ütemező modulhoz használt osztályok: szükség van a munkák leírásához használt osztályokra, az erőforrások leírásához használt osztályokra. A szerver komponenseihez olyan osztályokra van szükség ami egy kiszolgáltatott portot nyit meg, hívásokat fogad azokon a portokon, valamint kapcsolatot bont. A kliens komponens osztályai közé pedig olyan osztályok tartoznak, melyek kapcsolatot létesítenek a szerverrel, kéréseket küldenek a megfelelő paraméterekkel.A további klienshez szükséges osztályok a gráf feldolgozásához, és eredmény megjelenítéséhez kellenek.

5. Szoftver struktúrájának megvalósítására való probálkozások

5.1 Az ütemező modul DLL – be való fordítása

Az eredeti elképzelés a megvalósításra: a szerver programjára az volt, hogy C# környezetben és programnyelvvel valósítsam meg. Mivel az ütemező modul C++ nyelven íródott és a terjedelme elég nagy volt, ezért ahhoz, hogy elkerüljem a C++ nyelveben írt kód nagyobb módosításait és ne az egészet kelljen újra átírni, átdolgozni, ezért a C++ kódot úgynevezet DLL formátumba fordítottam volna le, hogy a C# programozási nyelv felhasználhassa, mint egy beimportált könyvtárat. Ezen megoldás egy köztes modult

eredményezett volna, ami növelte volna az alkalmazás rétegezettségét. A megoldás ezen verziója programozása során viszont számos problémába ütköztem. Elsőként a C++ kód fordításának megváltoztatása merült fel. A fordítás során egy fájllal többet kellett létrehoznom, amit a DLL másik nevén alkalmazáskiterjesztés meghatározott metódusainak leírására szolgált. A DLL - be való fordításra a választásom a GCC (Gnu Compiler Collection) – ra esett. A GCC egy ingyenes nyílt forráskódú fordító, mely több millió C programkód sorból és több ezer kapcsolóból áll. Rengeteg programnyelv fordítására alkalmas, mint például a C, C++, és a Fortran. A GCC fordításhoz szükségem volt egy úgynevezett makefile – ra. A makefile – ra azért volt szükségem, mert a cmake rendszert használtam. A GCC fordító szerencsére használható, mind Windows, mind Linux alapú operációs rendszerekben. A Cross Platform tulajdonsága rendkívül hasznos, mikor Windows segítségével szeretném fejleszteni a programot, de Linux operációs rendszer alatt szeretném lefuttatni a forráskódot. A GCC további előnyei még, hogy a DLL elkészítéséhez csak a kapcsolókon kell változtatnom, hogy a forráskódot ne futtatható állományként képezze le, hanem mint alkalmazáskiterjesztést. Ezen feladat megoldható a GCC által használt kapcsolókkal. A hozzáadott egy fájl a függvények felhasználható formáit határozta meg. Ezek a függvények interfész szerűen definiciós és implementációs módszerekkel kerültek megvalósításra. A definíciós rész a declspec kulcsszóval történik ezen esetben. A DLL-es megoldás metódusaiból következően a paraméterek átadása és a két nyelv közötti közös adattípusok jelentették az egyik problémát. A legtöbb adattípus a két programnyelvben megegyezik, viszont ami a legnagyobb különbség, az az, hogy a C++ programnyelv a tömbök kezelésére a mutatókat használja. A C# pedig csak az unsafe mód engedélyezése után teszi lehetővé a tömbök használatát. A két nyelv máshogyan kezeli ezen változók memóriafoglalását. Erre azzal a megoldással álltam elő, hogy a C++ oldalon a mutatót használom, még a C# oldalon az egyszerű tömböt. Ezen feladat megvalósítása felesleges elkerülhető lépéseket generált volna, mivel a C++ egy külön paramétert használ, amivel a tömb méretét jelzi. A C# tömb objektuma automatikusan számolja ki és tárolja a tömb méretét, nem kell számon tartani külön változóval. Ezen külön változó egy felesleges elem lenne a C# programnyelv részéről. A C++ programnyelv részéről viszont kötelező lenne. Még megoldandó feladatot jelent a komplex vagyis összetett adatstruktúrák közötti kapcsolat megvalósítása. Az osztályok ezen a téren nem jöhettek szóba, mivel a két nyelv osztályai és azok példányai jelentősen különböznek. Emiatt a komplex változókkal való kommunikáció megoldásához a strukturákhoz fordultam. A struktúrák az osztályokkal ellentétben nagyon hasonlóak a két nyelvben. A struktúrák majdhogynem teljesen megegyeznek. A mutató és tömbök problémái itt is megjelennek, viszont erre megoldást a

Microsoft által biztosított könyvtárak segítik. A strukturák belső adattagjainak összekapcsolását pár sor programkóddal megoldhatjuk melyeket a belső adattagok felé elhelyezve a programkódban rákényszeríthetjük, hogy C++ kompatibilisek legyenek. Ezzel a problémám az volt, hogy a hozzáadott tömb méretét jelző paraméterek problémája szintén előjön. A C++ oldalon a feldolgozó függvények belsejébe ez a lépés felesleges for ciklusokat generál, hiszen az átadott adatokat amit C# programnyelvvel kapcsolatban juttattunk el, a C++ oldalon fel kell tudnunk dolgoztatni. A feldolgoztatáshoz pedig szükséges az extra paraméter vagy változó. A megoldást követve a rengeteg ciklus és változók átadása jelentősen megnövelte a módszer bonyolultságát és átláthatóságát. Mialatt ezt a módszert alkalmaztam a leggyakoribb probléma, amibe ütköztem, az a read access violation C++ vagy C hibakód volt. A felhasználandó paraméterek összekötésére még a JSON szöveges változó is felmerült, viszont ez egy felesleges konverzióval járt volna, amit a C++ nyelveben akkor az idő szűke miatt nem tudtam befejezni, idővel a JSON formátumra egy megoldással elő kellett állnom, mivel minden világhálón történő kommunikáció a JSON formátumot használja. Ennek megoldását egy másik fejezetben részletezem. Összeségében a növekedő komplexitás és a vele járó hosszú programkód a réteg ötletének eltörlését eredményezte.

5.2 A cpprest microsoft által készített könyvtár használata:

Miután már eldöntöttem, hogy csak két réteg lesz az alkalmazásban: a szerver és a kliens, a köztes réteget eltörlöm. Áttértem a C++ szerver oldali fejlesztésére. Az elképzelés mostmár tisztán C++, ennek használatával szerettem volna megvalósítani a szerver alkalmazásrészt. Az első könyvtár amivel próbálkoztam a cpprest könyvtár volt, amit a Microsoft készített. A cpprest könyvtár egy nyílt forráskódú C++ könyvtár, mely portok megnyitására, és http kérések befogadására alkalmas. Ezen megoldás működőképesnek látszott hosszú ideig. A kéréseket képes volt feldolgozni, meg is tudta válaszolni. A probléma viszont akkor keletkezett, mikor böngészőt használó klienssel próbáltam összekötni. Ekkor a CORS (Cross Origrin Resource Sharing) mechanizmus blokkolta a szerver válaszát a böngésző felé. Erre a problémára a mai napig nem találtam megoldást sehol, mivel a könyvtár dokumentációja nem elég részletes, ezért a több tízezer soros programkódban kell keresnem egy módot, hogy megoldjam. Ez nem bizonyult járható útnak az idő szűke miatt, nem tudtam volna befejezni a szakdolgozat kiírt ideéig.

5.3 A kommunikációhoz használt kliens osztály JavaScript-ben való megvalósítása

Nem csak a szerver oldalon keletkeztek problémák, mikor kapcsolatot próbáltam létesíteni a két komponens között. A kliens oldalon a fájlok kezelésére először a C# kód másféle kódolást alkamazott, mint amilyenre a szervernek szüksége volt, ezért a szerver nem volt képes feldolgozni a beérkező szöveget. Ha ugyanazon szöveget a Postman nevű programból küldtem el a szerver felé, és annak az üzenetnek a kódolása megfelelő volt. Innen kaptam az ötletet, hogy mivel a postman javascript programnyelvet használ, ezért a kliens alkalmazásomba javascript segítségével küldöm el a szöveget az adatokról. A javascript – es megoldás nem bizonyult hatásosnak, mivel a C# által meghívott javascript ugyanúgy megváltoztatta a beolvasott szöveg kódolását. A végső megoldás pedig a C# által biztosított beolvasás paramétereinek módosítása lett.

5.4 A felhőben való elhelyezés virtuális konténerizáció nélkül

Az elkészült alkalmazást szükséges futtatni egy számítógépen belül. Ezen cél érdekében én a tanszéki linux számítógépek egyikét választottam. Miután meggyőződtem arról, hogy a szerver komponense az alkalmazásnak elkészült és futásra képes a tanszéki számítógépen a WinSCP program segítségével töltöttem fel a forráskódot, majd ott állítottam össze a fejlesztőkörnyezetet a futtatáshoz. A GCC fordító azon a számítógépen sajnos nem volt képes a futtatható állományt előállítani, mert a beimportált könyvtárakat nem találta meg. A tisztán a tanszéki számítógépes futtatás ötletét így elvetettem és egy Docker nevezetű konténerizációs programot telepítettem, és az alkalmazást egy úgynevezett docker container – ben futtatom.

5.5 A végpontok közötti átirányítás nginx használatával

A két komponens közötti kommunikációhoz egy köztes szerver kezelő szoftvert szerettem volna elhelyezni, aminek a neve nginx. Ezen ötletemet elvetettem, mivel az nginx konfigurációja átalakította az általam beállított http kommunikáció fejléceit így a CORS hiba megint megjelent. Az nginx - szel kapcsolatos problémákból következően az nginx használatát elvetettem.

6. A rendszer implementálása

6.1 A konténerizáció részletei

Az alkalmazás konténerizálásához a Docker nyílt forráskódú programot választottam. A konténerizáció lényegében egy virtuális gép létrehozását jelenti, melyben alkalmazásokat futtathatunk anélkül, hogy a gazda számítógépre telepítettünk volna bármilyen fejlesztő környezetet. A fejlesztőkörnyezetet továbbá is telepíteni szükséges, viszont azokat a konténeren belül kell. A konténerre szükséges a forráskód felmásolása, anélkül nem tudnánk futtatni. Ha az alkalmazás valamilyen nagy károsodáshoz vezető hibába érkezne, akkor először a konténer omlik össze ezzel megvédve a gazda számítógépet. A docker konténer belső portjai, amiken keresztül alkalmazásokat tudunk megnyitni, függetlenek a gazdagép portjaitól. A gazdagép egyetlen portján keresztül érjük el a docker konténer publikussá tett portját, ezt a portot a konténer létrehozásánál kell megadnunk, azzal a porttal együtt, amelyiket felhasználja a konténerünk. A docker konténer minden esetben linux operációs rendszerrel rendelkezik. A létrehozásnál megadhatjuk a linux disztribúció típusát és az előre feltelepített programok és fejlesztőkörnyezetek listáját. Az általam használt konténer disztribúciójának az ubuntu – t választottam. Alapesetben debian disztribúciójú a konténer. Az ubuntu – t azért választottam, mert az apt csomagkezelő mellett a snap csomagkezelővel is rendelkezik. Ezen lépéssel több általam használt csomag használatát biztosítottam. A szoftver amit a snap csomagkezelőből kellett telepítenem a gh alkalmazás volt. A gh a github verziókezelő használatához volt szükséges, hogy a szerver által használt könyvtárakat le tudjam tölteni és importálni. A forráskódomat a gazdagépről másoltam fel. A makefile include mezőit is meg kellett változtatnom, mivel más elérési útvonalakat tartalmazott, mint a gazdagépé. A konténeren belül való GCC fordító futtatása már lefordította a forráskódomat, ellentétben a gazdagéppel való

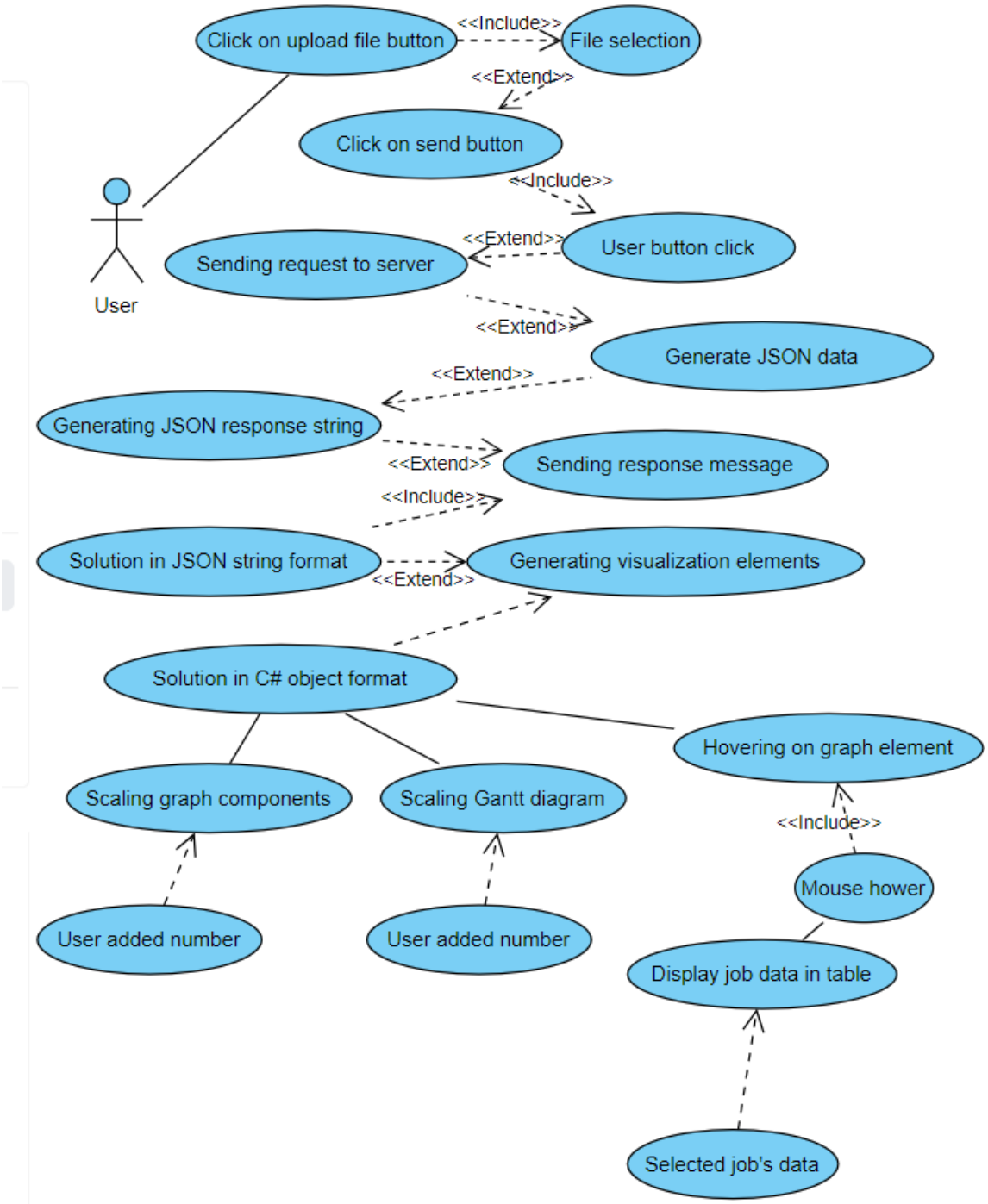
használattal. A szerver futtatható állományának elkészítése után a futtatás parancs kiadása után a szerver komponens már elérhető egy böngészőből az alábbi hiperhivatkozáson: http://compaq.iit.uni-miskolc.hu:3008 a kliens komponens ezen a hivatkozáson kommunikál a szerverrel.

6.1.1 A konténerizáció alapvető karbantartási parancsai

A docker konténer karbantartásához rengeteg konfigurációs beállítást használhatunk. Ebben a fejezetben az általam használt parancsokat szeretném ismertetni. Az első a konténer létrehozása, amit a docker run paranccsal tudunk megtenni. A hozzá tartozó kapcsolókból is több fajta van. Azok amelyeket használtam a -t a terminál fenntartás parancsa, a -p amivel a kiosztott hálózati portot határozzuk meg, a bash kapcsolóval pedig az automatizáló programok futtatási módját határozzuk meg, a konténernek image – ét egy szóközzel megadott linux disztribúció neve adja. A disztribúciók közül is rengeteg választékot találunk, ha üresen hagyjuk ezt a mezőt, akkor a Debian az alapértelmezett linux disztribúció. Én az Ubuntu nevezetű disztribúciót választottam, ami a Debian disztribúció egyik kiterjesztése. Azért az Ubuntu – t választottam és nem a Debian – t, mert a Debian csak az apt csomagkezelőt tartalmazza, az Ubuntu pedig a snap csomagkezelőt is tartalmazza, ennek köszönhetően a snap csomagkezelőből több féle alkalmazáshoz van hozzáférésem, a csomagok amelyekre szükségem volt, a gh ami a github kezeléséért felelős. A github weboldalról további nyílt forráskódú csomagokat töltöttem le, mint az asio és a cppcrow, a cppcrow a C++ szerverhez volt szükséges, az asio pedig a cppcrow – hoz volt szükséges. Ezeket a git clone paranccsal töltöttem le. Mielőtt ezeket a projekteket le tudnánk tölteni egy github autentikáció szükséges, amiért a gh alkalmazás felelt. A létrehozás legutolsó paramétere pedig az előre feltelepített csomagok és alkalmazások paramétere. A docker nem futtatható linux, alatt csak az úgynevezett root vagyis gyökér felhasználóként, amit a sudo kulcsszóval érünk el. Az én programom esetében a gcc fordító alkalmazásra volt szükségem. Ezen körvolnalazások után a létrehozás parancsa a következő képpen írható le: sudo docker run -t -p 3008:18080 ubuntu gcc bash parancs. Miután ez lefutott, a konténerünk elindul, vagy legalábbis megprobál, mert ha valamilyen paraméter nem helyes, akkor elindul és azt követően azonnal leáll. Ha a megadott hálózati port foglalt, akkor a docker nem futtatja a létrehozó parancsot, hibaüzenetként jelzi ki, hogy az általunk kiválasztott port már foglalt, ezt a viselkedést egy másik futó alkalmazás vagy esetleg egy nginx konfiguráció is kiválthatja. A konténer elindulását követően a programunkat is fel kell töltenünk rá, hiszen ha a konténer nem tud az alkalmazásunkról, akkor futtatni sem

tudja. Az alkalmazást feltöltés után el is kell indítanunk a konténerben, ezen lépést megelőzi a könyvtárak letöltése, amit viszont a github telepítése előz meg. Ha a könyvtárak megvannak, akkor a gcc fordítóhoz megadott makefile – t kell megváltoztatnunk. A feltöltött kódban a makefile import paraméterei a gazdagépre vannak formálva eredetileg. Az import paramétereket át kell írnunk a feltöltött makefile – ban, mivel a konténer másfajta elérési útvonalakkal dolgozik. Ha ezekkel megvagyunk, akkor kiadhatjuk a make parancsot, ami elkezdi a forráskód lefordítását, ha az hibátlanul lefutott, akkor elindul létrejön az alkalmazásunk. Az alkalmazás kap egy nevet, amelyre hivatkozhatunk. A hivatkozási név segítségével indíthatjuk el az alkalmazást vagyis a programot. Az alkalmazás egészen addig fut ameddig, le nem állítjuk, vagy egy olyan hibába ütközik, amely következtében az alkalmazás összeomlik. Ha folyamatosan szeretnénk futtatni, akkor nyitva kell tartanunk azt az ablakot és terminált, amivel elindítottuk. Ez főként tesztüzemmódban előnyös, mert folyamatosan nyomon tudjuk követni a szerver működését. Üzemszerű használathoz célszerű script-et használni, amely elindítja a konténerben az alkalmazást. A konténert leállíthatjuk a sudo docker container stop NÉV/ID paranccsal. A név vagy az id a docker konténer azonosításához szükséges. A név megadása nem kötelező az id viszont igen, A parancsok esetében az id – vel és a névvel is beazonosíthatunk egy konténert, ezért lehet a stop parancsnál az id vagy a név alapján azonosítani. Alapesetben egy konténernek egy véletlenszerűen kiosztott nevet ad a Docker. A konténer nevét a létrehozásnál is megadhatjuk a --name kapcsolóval. A konténernevet időközben is megváltoztathatjuk, a sudo docker rename paranccsal. Módosítani egy konténeren csak leállított esetben lehet. Konténerek létrehozása és módosítása mellett még törölhetjük is a megállított konténereket a sudo docker container remove NÉV/ID paranccsal. Amennyiben a parancs kiadása alatt a konténer fut a program egy hibaüzenetet ír, azzal az üzenettel, hogy törölni futás alatt lévő konténert nem tud. Ha mégis ki akarjuk mindenáron törölni a konténert, még futás alatt, akkor ezen műveletet a -- force kapcsolóval tehetjük meg. A force kapcsoló végrehajtja helyettünk a leállítást. Miután egy konténert leállítottunk, de újra szeretnénk használni, elindíthatjuk a sudo docker container start NÉV/ID paranccsal. Ha minden rendben talált a docker, akkor elindul a konténer és már a terminájába is beléphetünk. A terminálba belépés paranccsa a sudo docker container exec -it NÉV/ID bash.

6.2 A szoftver use case diagramja



12. ábra

Amit a felhasználó lát a felületen az a kliens által legenerált elemek halmaza. A weboldal elején két darab gomb található, az egyik a fájl kiválasztásáért felelős, amit a felhasználó a programnak, mint paraméter fog megadni, ahogyan az ábrán is látható, bemeneti paraméterként a fájlt jelöltem meg. Miután a felhasználó kiválasztotta a kérdéses fájlt utána, ha a másik gombra: a küldés gombra kattint a szerver felé továbbítja a kérést. A küldés gomb

bemenete igazából a felhasználó által jelzett kattintás. A kattintás után történik az adatok JSON szöveggé való konvertálása, ami az elküldött szöveg paramétere lesz. Az elküldést követően a szerver modul elkészíti a megfelelő C++ osztálypéldányt, majd az osztálypéldányból előállítja a megoldást. Az elkészült megoldást JSON formátumba konvertáltatjuk, aztán a szerver visszaküldi a megoldást. A JSON formátumú megoldás a visszaküldött üzenet paramétere. A beérkező válaszból C# objektumot készít a kliens. Az elkészült objektumból a kliens ezután legenerálja a további HTML elemeket. A legenerált elemek több felhasználói interakciót biztosítanak. Az első a gantt diagram. A gantt diagram bemeneti paramétere a felhasználó által megadott skálázhatóság egy bemeneti mezőben. A második a gráf megjelenítése. A gráf ugyanúgy skálázható, mint a gantt diagram bemeneti paramétere szintén a skálázást módosítja. A gráfnak viszont több féle komponensét lehet skálázni: a munkákat jelző körök sugarát skálázhatjuk, a kapcsolatot összekötő vonalak vastagságát skálázhatjuk, továbbá a nyilak méretét is skálázhatjuk. Ha a gráf kapcsoló vonalaira viszi az egerét a felhasználó egy táblázat jelenik meg alatta, mely a hozzá tartozó munka adatait mutatja be. A munkához tartozó adatok a következők: a munkára rákövetkező másik munka, illetve munkák. Az adott munka időtartama. Ezen akció bemeneti paramétere értelem szerűen a kiválasztott munkához tartozó objektum, amely tartalmazza a munkához tartozó adatokat. Minden utólag generált felületei elem által biztosított cselekmény megismételhető. A gantt diagram skálázhatósága megváltoztatható. A gráf elemeit átméretezhetjük tetszésünk szerint, valamint új munka kiválasztása esetén a táblázat maga is újragenerálódik, és újrarajzolódik.

6.3 Kódrészletek

6.3.1 A szerver megvalósítás kódrészletei

A struktúra megvalósítás próbálkozásainál lévő problémákat sikerült megoldanom az alábbi kódrészletekben. Elsőként a cppcrow segítségével megvalósított szerver kódrészleteit szeretném bemutatni. A kód tartalmazza a port felnyitását és a szerver elindítását, mint C++ crow egyszerű alkalmazást, JSON szövegként kapott információk kifejtését és objektummá alakítását, valamint a válasz objektum JSON szöveggé való konverziót. Tizenharmadik ábra, tizennegyedik ábra, tizenötödik ábra, és tizenhatodik ábra.

```
// CppCrow.cpp : This file contains the 'main' function. Program execution begins and ends there.
//
#include <crow.h>
#include <iostream>
#include "../../../vcpkg/vcpkg/packages/crow_x64-windows/include/crow/middlewares/cors.h"
#include "PJ_MS.hpp"
#include "PJ_SYS.hpp"
#include <nlohmann/json.hpp>


int main()
{
    // Enable CORS
    crow::App<crow::CORSHandler> app;

    auto& cors = app.get_middleware<crow::CORSHandler>();
    cors.global().origin("*").allow_credentials().headers("Accept", "Origin", "Content-Type",
        "Authorization", "Refresh", "mode", "Access-Control-Allow-Methods", "Allow",
        "Access-Control-Allow-Origin", "Access-Control-Allow-Crerdentials"
    ).methods(
        crow::HTTPMethod::GET, crow::HTTPMethod::POST, crow::HTTPMethod::OPTIONS, "POST"_method, "GET"_method,
        crow::HTTPMethod::HEAD, crow::HTTPMethod::PUT, crow::HTTPMethod::Get, crow::HTTPMethod::Post
    );

   CROW_ROUTE(app, "/add_json")
       .methods(crow::HTTPMethod::POST)
       ([](const crow::request& req) {

       crow::json::rvalue Data = crow::json::load(req.body);
       nlohmann::json nData = nlohmann::json{ {req.body} };

       std::string mydata = Data["Data"].s();
       //std::cout << mydata.c_str();
       if (!Data) {
           crow::response badresponse = crow::response(crow::status::BAD_REQUEST);
           badresponse.add_header("Allow", "*");

           return badresponse; // same as crow::response(400)
       }
```

13. ábra

```
nlohmann::json response;
std::ostringstream os;
//crow::json::wvalue response = new crow::json::wvalue();
PJ_SYS pjs;
char out_ext1[] = "_sch_result.txt";
int K = 6;   //number of objective functions
double* w;   //priorities

int readResult = pjs.read_PSPLIB_inputdata_from_string(mydata);
//readResult = 1;

switch (readResult)
{
case 1:
    w = new double[K];
    //They are currently not active
    w[0] = 0;   //Lmax
    w[1] = 0;   //Tmax
    w[2] = 0;   //Tsum
    w[3] = 0;   //Usum
    w[4] = 1;   //Cmax
    w[5] = 0;   //Csum
    pjs.run(K, w, out_ext1);
    delete[] w;
    response = pjs.GetResultResponse();
    break;
case 2:
    response = { {"message", "Number of resources cannot be lower than 0."} };
    break;
case 3:
    response = { {"message", "Resource innitial resource capacity cannot be lower than 0."} };
    break;
case 4:
    response = { {"message", "Job id cannot be greater than the number of jobs, or lower than 1"} };
    break;
case 5:
    response = { {"message", "Invalid successor number."} };
    break;
case 6:
    response = { {"message", "A successor cannot point to a non existed job"} };
    break;
case 7:
    response = { {"message", "Number of jobs must be lower than 0."} };
    break;
case 8:
    response = { {"message", "Required resource must be greater then 0."} };
    break;
case 9:
    response = { {"message", "Successor identification cannot be higher than the jobs' number" } };
    break;
default:
    //Fix later
    response = { "message", "Unhandled error accured." };
    break;
}
auto resultresponse = crow::response{ response.dump(4) };
resultresponse.add_header("Allow", "GET,POST,OPTIONS");
return resultresponse;

    });

CROW_ROUTE(app, "/")([]() {
auto resultresponse = crow::response{ "Hello world" };
resultresponse.add_header("Allow", "*");

return resultresponse;
});

    app.bindaddr("127.0.0.1").port(18080).multithreaded().run();
}
```

14. ábra

```
nlohmann::json PJ_SYS::GetResultResponse()
{

    //Resource capacity
    nlohmann::json rcs = {
            0
    };

    rcs.push_back(0);

    nlohmann::json times = {
        0
    };

    times.push_back(0);

    nlohmann::json resS = {
        {
            {
                "RC",
                rcs
            },
            {
                "Time",
                times
            },
            {
                "Init_RC",
                0
            }
        }
    };

    for (int i = 0; i < m_PJMS->get_NR(); i++)
    {
        resS.push_back(
        {
                {
                    "RC",
                    rcs
                },
                {
                    "Time",
                    times
                },
                {
                    "Init_RC",
                    0
                }
         }
        );
    }

    nlohmann::json jobs;
```

15. ábra

```
//The first item is empty
for (int i = 0; i < m_PJMS->get_NJ(); i++)
{
    std::vector<int> sucList = std::vector<int>();
    for (int j = 0; j < m_PJMS->get_Job()[i].get_NSuc(); j++)
    {
        sucList.push_back(m_PJMS->get_Job()[i].get_Suc_i(j + 1));
    }
    std::vector<int> Prec = std::vector<int>();

    for (int j = 0; j < m_PJMS->get_Job()[i].get_NPrec(); j++)
    {
        Prec.push_back(m_PJMS->get_Job()[i].get_Prec_i(j+1));
    }
    //overrides data if it has the same name.
    nlohmann::json job;

    job["id"] = m_PJMS->get_Job()[i].get_ID();
    job["ProjectID"] = m_PJMS->get_Job()[i].get_ProjectID();
    job["ProcT"] = m_PJMS->get_Job()[i].get_ProcT();

    job["Suc"] = sucList;
    job["Res"] = *m_PJMS->get_Job()[i].get_Res();
    job["Req"] = *m_PJMS->get_Job()[i].get_Req();
    job["DueDate"] = m_PJMS->get_Job()[i].get_DueDate();
    job["NPrec"] = m_PJMS->get_Job()[i].get_NPrec();
    job["Prec"] = Prec;
    job["NSuc"] = m_PJMS->get_Job()[i].get_NSuc();
    job["EST"] = m_PJMS->get_Job()[i].get_EST();
    job["EFT"] = m_PJMS->get_Job()[i].get_EFT();
    job["LST"] = m_PJMS->get_Job()[i].get_LST();
    job["LFT"] = m_PJMS->get_Job()[i].get_LFT();
    job["ST"] = m_PJMS->get_Job()[i].get_ST();
    job["ET"] = m_PJMS->get_Job()[i].get_ET();

    jobs.push_back(job);
}

nlohmann::json response;
//Hardcoded values here, change later
response = { {"Lmax", 0}, {"Tmax", 0}, {"Tsum", 0},{"Usum", 0}, {"Cmax", 1}, {"Csum", 0},
    {"NJ", m_PJMS->get_NJ()}, {"NR", m_PJMS->get_NR()},{"jobs", jobs},{"res",resS}};


return response;
```

16.ábra

6.3.2 A kliens megvalósítás kódrészletei

Mint már említettem a kliens a felhasználó által látható elemek megjelenítéséért felel elsősorban, de inkább a kliensen belüli kliens objektumot mutatnám be, ami az eredmények feldolgozásához szükséges. Ennek a feladatnak az egyik eleme, hogy a JSON szövegből egy objektumot készítsen. A későbbiekben mutatok kódrészletet a felületi elemek kirajzoltatásának kódjára is. Tizenhetedik ábra, tizennyolcadik ábra.

```
public async Task SendRequest()
{
    string contentString = "";
    Http.DefaultRequestHeaders.Add("Access-Control-Allow-Origin", "http://localhost:18080");
    Http.DefaultRequestHeaders.Add("Access-Control-Allow-Crerdentials", "200");
    Http.DefaultRequestHeaders.Add("Accept", "*/*");
    Http.DefaultRequestHeaders.Add("mode", "no-cors");


    //content.Headers.Add("Content-Type", "application/json");

    Response response = new Response();
    response.Data = FileData!;
    contentString = JsonSerializer.Serialize(response);
    //contentString = JsonSerializer.Serialize(response);
    //var message = await Http.PostAsync("http://127.0.0.1:18080/add_json", new StringContent(contentString));
    StringContent stringContent = new StringContent(contentString);
    //stringContent.Headers.Add("Allow","*");
    //IP on docker: 172.17.0.3:18080/add_json
    //IP on university server: http://compaq.iit.uni-miskolc.hu:8080/add_json
    //IP normal: http://127.0.0.1:18080/add_json
    //var message = await Http.PostAsync("http://compaq.iit.uni-miskolc.hu:8080/add_json", stringContent);
    HttpClient httpClient = new HttpClient();
    var message = await httpClient.PostAsync("http://compaq.iit.uni-miskolc.hu:8080/add_json", stringContent);


    //FileData = FileData!.Replace(System.Environment.NewLine,"@");

    string GetUrl = $"http://127.0.0.1:18080/test?Data=\"{FileData}\"";
    //Console.WriteLine(FileData.Contains(System.Environment.NewLine));
    Console.WriteLine(GetUrl);

    //var message = await Http.GetAsync(GetUrl);


    myAPIresponse = await message.Content.ReadAsStringAsync();
    try
    {

        if (myAPIresponse.Contains("message"))
        {
            var error = JsonSerializer.Deserialize<PJerror>(myAPIresponse, new JsonSerializerOptions() { });
            Console.WriteLine(error);
        }
        else
        {
            //Console.WriteLine(myAPIresponse);
            testData = JsonSerializer.Deserialize<Jdata>(myAPIresponse, new JsonSerializerOptions() { IncludeFields = true })!;
            //Some properties return with really bad results


            targetfunctions = new List<double>();

            targetfunctions.Add(testData.Lmax);
            targetfunctions.Add(testData.Tmax);
            targetfunctions.Add(testData.Tsum);
            targetfunctions.Add(testData.Usum);
            targetfunctions.Add(testData.Cmax);
            targetfunctions.Add(testData.Csum);

            int[] successorArray;


            graphInfos = new List<GraphInfo>();
```

17. ábra

```
            for(int k=1; k< testData.jobs.Count;k++)
            {

                GraphInfo info = new GraphInfo();
                info.Successors = new List<int>();
                JustArrays sucarray = new JustArrays();
                sucarray.id = k;
                successorArray = testData.jobs[k].Suc.ToArray();
                sucarray.firstElement = successorArray[0];
                SucArrays.Add(sucarray);

                if (successorArray.Length != 0)
                {
                    // info.Successors.Add(successorArray[0]);
                    for (int i = 1; i < successorArray.Length; i++)
                    {
                        info.Successors.Add(successorArray[i]);

                    }
                }

                info.id = testData.jobs[k].id;
                graphInfos.Add(info);
                if (k==testData.jobs.Count)
                {
                    break;
                }
            }

            Copied = testData;
            ShowCalculations = true;
        }


    }

    catch (Exception exc)
    {
        Console.WriteLine("Outer exception");
        Console.WriteLine(exc.Message);
    }

}
```

18. ábra

A következő kódrészletek a gantt diagram megjelenítéséért felelősek. Tizenkilencedik ábra, huszadik ábra.

```
@if (ShowCalculations)
{
    <MudNumericField Style="width:25%" @bind-Value="ganntScale"
Label="Scale of the gantt diagram:" Variant="Variant.Outlined" Min="1"/>
    <!-- border:1px solid black; -->
    <div width="100%"  style="  overflow-x:auto" >
        <h2>
            The gannt diagram of the jobs:
        </h2>
<svg width=@DataToString(GanntWidth() * ganntScale + 45)
 height="@DataToString(testData!.jobs.Count * 34)" >
        @for (int i = 1; i < testData!.jobs.Count; i++)
        {
        <rect fill="yellow"
                x=@DataToString(testData.jobs[i].ST * ganntScale + 25)
              y=@DataToString((i-1) * 33)
                 width=@DataToString((testData.jobs[i].ET -
                testData.jobs[i].ST)*ganntScale)
              height="30" stroke="cyan" stroke-width="2" />
```

19. ábra

```
            <svg>
                <text x="5" y=@DataToString((i) * 33)
                      fill="blue"> @DataToString(i) </text>
            </svg>
        }
        <svg>
           <line x1="23" y1=@DataToString((testData!.jobs.Count * 32) )
             x2="23" y2=20 fill="blue" stroke-width="2"
           stroke="blue" />
        </svg>
        <svg>
            <line x1="25" y1=@DataToString(testData!.jobs.Count * 32)
             x2=@DataToString(25 + GanntWidth() * ganntScale)
            y2=@DataToString(testData!.jobs.Count * 32) fill="blue"
            stroke-width="2" stroke="blue" />
        </svg>
        @for (int i = 0; i < GanntWidth(); i++)
            {
                if (i % 20 == 0 || i == GanntWidth())
                {
                    <svg>
                        <text fill="blue" x=@DataToString(i * ganntScale + 25)
                     y=@DataToString(testData!.jobs.Count * 32.5 )>
                            @i
                        </text>
                    </svg>
                }
            }

    </svg>
</div>
```

20. ábra

A precedencia gráf elemeit megjelenítő kódrészlettel folytatnám. Huszonegyedik ábra, huszonkettedik ábra, huszonharmadik ábra, huszonnegyedik, huszonötödik ábra.

```
<div>
    <div style="flex-flow:row; display:flex; ">

<MudNumericField  @bind-Value="circleRadius" Label="Radius of the circles:"
    Variant="Variant.Outlined" Min="1"/>
<MudNumericField  @bind-Value="lineSize" Label="Stroke of the lines:"
     Variant="Variant.Outlined" Min="1"/>
<MudNumericField  @bind-Value="arrowSize" Label="Size of the arrows:"
     Variant="Variant.Outlined" Min="1"/>

    </div>

<div width="100%"  @onmouseenter=@(e=>{
})  @onmouseup=@(e => {
    if (MoveCircle)
            {
 selected = Copied.jobs.Where(gi => gi.id == Howered ).FirstOrDefault()!;
 selected.positions[0] += e.ClientX - X;
 selected.positions[1] += e.ClientY - Y;
            }
 }) @onmouseleave=@(e => {
     selected = new Job();
     MoveCircle = false;
 })>
```

21. ábra

```
<svg width="100%" height="400">
     <defs>
        <marker id="arrow" refX=@DataToString(9) refY=@DataToString(3)
             markerWidth=@DataToString(arrowSize)
            markerHeight=@DataToString(arrowSize)  orient="auto-start-reverse"
            viewBox="0 0 10 10">
            <path d="M 0 0 L @DataToString(6) @DataToString(3) L 0
             @DataToString(6) z"/>
        </marker>
    </defs>

    @foreach (var gInfo in Copied!.jobs)
    {
        //&& gInfo.id < Copied.jobs.Count -3
        if (gInfo.id > 0 )
        {
            foreach (int successor in gInfo.Suc)
        {
            if (successor < Copied.jobs.Count)
                            {
                    //It is not in itself
                        <line x1="@gInfo.positions[0]"
                    y1="@gInfo.positions[1]"
                        @onmouseenter=@(e => {
                HoweredPrecedence.RemoveAll(Removable);
                HoweredPrecedence.Add(gInfo.id);
                HoweredPrecedence.Add(successor);
                showSelected = true;
                if (!(HoweredPrecedence.Count < 2))
                {
                    selectedJob.RemoveAll(Removable);
                    selectedJob.Add(Copied!.jobs.Where(x => x.id ==
                     HoweredPrecedence[0]).FirstOrDefault()!);
                    selectedSuccessors.RemoveAll(Removable);
                    foreach (var item in selectedJob[0].Suc) {
                     selectedSuccessors.Add(item); }
                    selectedGraph.RemoveAll(Removable);
                    selectedGraph.Add(Copied.jobs!.Where(x => x.id ==
                     HoweredPrecedence[1]).FirstOrDefault()!);
                }

        })
         x2="@testData.jobs[successor].positions[0]"
        y2="@testData.jobs[successor].positions[1]" fill="red" stroke="red"
         stroke-width=@DataToString(lineSize) marker-end="url(#arrow)"
         />

                        }

    }
    }

}
```

22. ábra

```
    @foreach (var item in testData.jobs)
    {
        if (item.id > 0)
        {
        <circle fill="blue" cx="@item.positions[0]" cy="@item.positions[1]"
             r=@DataToString(circleRadius) @onmouseenter=@( e => {
            MoveCircle = true;
            X = e.ClientX; Y = e.ClientY; Howered = item.id;
            Console.WriteLine($" Howered {Howered}"); })  />
        }

    }

</svg>
</div>
@if (showSelected)
{
<h3>
        Job data:
</h3>
    <MudTable Items="@selectedJob.Take(2)">
        <HeaderContent>
            <MudTh>
                id:
            </MudTh>
            <MudTh>
                projectid:
            </MudTh>
            <MudTh>
                productiontime:
            </MudTh>
            <MudTh>
                resources:
            </MudTh>
            <MudTh>
                 required:
            </MudTh>
            <MudTh>
                duedate:
            </MudTh>
            <MudTh>
                EST:
            </MudTh>
            <MudTh>
                EFT:
            </MudTh>
            <MudTh>
                LST:
            </MudTh>
```

23. ábra

```
            <MudTh>
            LFT:
            </MudTh>
           <MudTh>
            ST:
           </MudTh>
           <MudTh>
            ET:
           </MudTh>
    </HeaderContent>
    <RowTemplate>
        <MudTd DataLabel="Id">
            @context.id
        </MudTd DataLabel="Productiontime">
        <MudTd>
            @context.ProjectID
        </MudTd>
        <MudTd>
            @context.ProcT
        </MudTd>
        <MudTd>
            @foreach (var item in context.Res)
            {
                @item
            }
        </MudTd>
        <MudTd>
            @foreach (var item in context.Req)
            {
                @item
            }
        </MudTd>
        <MudTd>
            @context.DueDate
        </MudTd>
        <MudTd>
            @context.EST
        </MudTd>
        <MudTd>
            @context.EST
        </MudTd>
        <MudTd>
            @context.EFT
        </MudTd>
        <MudTd>
            @context.LST
        </MudTd>
        <MudTd>
            @context.LFT
        </MudTd>
```

24. ábra

```
                <MudTd>
                    @context.ST
                </MudTd>
                <MudTd>
                    @context.ET
                </MudTd>
            </RowTemplate>
            <ChildRowContent>
                <MudTr>

                    <MudTh>
                        Successors:
                    </MudTh>
                    @foreach (int successor in context.Suc)
                    {
                        //There are negative successors which are bad indicating a bug
                        if(successor >= 0)
                        {
                            <MudTd>
                                @successor
                            </MudTd>
                        }
                    }

                </MudTr>
            </ChildRowContent>
        </MudTable>
    }
    else { }
</div>
}
else
{
    <div width="100%">

    </div>
}
</div>
<MudThemeProvider @ref="@_mudThemeProvider" @bind-IsDarkMode="@_isDarkMode"/>
```

25. ábra

7. Összefoglalás

Szakdolgozatomban egy kliens szerver koncepciójú webes alkalmazást fejlesztettem, amelybe integráltam egy meglévő ütemezésű modult. A dolgozat végigvezet a munkaütemezéshez szükséges alapfogalmakon. Az alapfogalmak nemcsak a munkaütemezéshez, hanem a projektütemezéshez is alkalmazhatók. A kettő között van különbség, de lényegében ugyanolyanok. Az alapfogalmak közé tartoznak a megjelenítő elemek, mint a Gantt diagram vagy a precedencia gráf. Nemcsak megjelenítő elemeket soroltam fel abban a fejezetben, hanem általánosabb ütemezéshez kapcsolódó definíciókat is, mint a késés, sietés vagy csúszás. Magyarázatot adok az ütemezéshez hozzátartozó célfüggvények jelölésére.

Kifejtem benne a feladatra kidolgozott általános szoftverstruktúrát. A legáltalánosabb struktúra a webes alkalmazások esetén a három főbb modul a szerver, a kliens, és az adatbázis. Mivel az én alkalmazásom nem rendelkezik sem felhasználókezeléssel, sem pedig elmentendő adatokkal, ezért csak a két legfontosabb modult hagytam meg a szervert és a klienst. Bevett séma továbbá a konténerizáció használata, mely lehetővé teszi az alkalmazások virtuális számítógépeken való futtatását. A szakdolgozatom kitér az általános Docker használati parancsokra is. A szerver megvalósításához a C++ programnyelvet, a kliens megvalósításához pedig a C# programnyelvet választottam. A keretrendszerek közül a C++ - hoz a CrowCpp – t és a gcc fordítót, a C# - hoz pedig a Microsoft által biztosított Blazor keretrendszert használtam.

Szót ejtek a megvalósítás kialakulásához nélkülözhetetlen próbálkozásokról. A próbálkozásokról, melyek között a programomat konténerizáció nélkül probálom futtatni, egy köztes réteget próbálok elhelyezni, a két modul közé, hogy máshogyan oldjam meg a kommunikációt, valamint az nginx konfigurációhoz való próbálkozásomat is.

Bemutatom a végleges szoftvert a kódjain és példáin keresztül. A végleges szoftver által biztosított felhasználói felületet egy úgynevezett use case diagramon keresztül szemléltetem. A kódokat két részre bontottam a szerver és a kliens kódrészletei. A szerverben kiemeltem a kommunikációt és a hibakezelést megvalósító modulokat.Magának az ütemezőnek a kódját nem részleteztem a szakdolgozatomban, hiszen az nem az én munkám. Mutatok példát egy http port megnyitására, hogy kéréseket fogadhasson a szerver, megjelenítő HTML elemek generálására, amelyeket értelmezve a böngésző megjeleníti a felületi elemeket, valamint a két rész kommunikációjára is, melyet JSON szöveg formátumban folytatnak egymással.

8. Summary

In my dissertation I integrated a job scheduling program modul into a webapplication. My dissertation walks you throught the basic definitions of job scheduling. Theese definitions are not only applicable for job scheduling but also for project scheduling as well. The two scheduling is different but they are essentially the same. The basic definitions contains the components, which appear, like the Gantt diagram or the precedence graph. I mentioned other common definitions linked to scheduling like lateness, tardiness or earliness. I also describe some functions, which are related to the scheduling problem.

I explain the usual solutions of the sofware structure, which is used for this problem. The most common structure for a webapplication is to set three parts: the server, the client, and the database. My application doesn't have user handling or something obligating any data saving so it doesn't require any databases, I only leave it with the two main parts the server and the client. Furthermore a common usecase is to containerize the application, which allows the application to run under a virtual computer. This document mentions the most common commands for using Docker. The server part was written in C++, for the client part I used the C# programming language. As for the frameworks in the case of C++ I used CrowCpp and the gcc compiler, for C# I used the Blazor framework provided by Microsoft.

I would like to mention some attampts for setting up this structure. The attempt without the containerization of the application, the attampts where I make a new layer between the two moduls to solve the communication between the two moduls, and where I tried an nginx configuration too.

I show the final software's codes and examples. The final software's user interface is shown throught a use case diagram. I separated the codes to two parts the server's and the client one's details. The schedulers code isn't at my dissertation because it wasn't written by me, so it isn't the part of my dissertation. I show an example how to open an http port for the server to accept requests, or to generate HTML components, which with the help of the browser makes the user interface visible, or the communication between the two parts, which stores the messages using JSON format.

9. Irodalomjegyzék

[1]. Kulcsár G.: Termelésinformatika alapjai, előadásvázlatok, Miskolci Egyetem, http://ait.iit.uni-miskolc.hu/~kulcsar/serv05.htm

[2]. Kulcsárné Forrai M.: Erőforrás tervezés, előadásvázlatok, Miskolci Egyetem, https://www.uni-miskolc.hu/~aitkfm/

[3]. Kulcsár G.: Diszkrét termelési folyamatok számítógépes tervezése és irányítása, előadásvázlatok, Miskolci Egyetem, http://ait.iit.uni-miskolc.hu/~kulcsar/serv01.htm

[4]. P. Brucker: Schedulig Algorithms, Springer, 1995.

[5]. M. L. Pinedo: Scheduling Theory, Algorithms and Systems, Springer, 2008.

[6]. Kulcsár G.: Virtuális Vállalt c. tantárgy előadásvázlatok, Miskolci Egyetem, http://ait.iit.uni-miskolc.hu/~kulcsar/serv04.htm/

[7]. A. A. B. Pritsker, W. J. Lawrence, and P. M. Wolfe, "Multiproject Scheduling with Limited Resources: A ZeroOne Programming Approach," Management Science, vol. 16, no. 1, pp. 93-108, Sept. 1969.

[8]. J. Blazewicz, J. K. Lenstra, and A. H. Kan, "Scheduling subject to resource constraints: classification and complexity," Discrete Applied Mathematics, vol. 5, no. 1, pp. 11-24, Jan. 1983, doi: 10.1016/0166-218X(83)90012-4.

[9]. S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," European Journal of Operational Research, vol. 207, no. 1, pp. 1-14, 2010, doi: 10.1016/j.ejor.2009.11.005.

[10]. K. Taha, "Methods That Optimize Multi-Objective Problems: A Survey and Experimental Evaluation," IEEE Access, vol. 8, pp. 80855-80878, 2020, doi: 10.1109/ ACCESS.2020.2989219.

[11]. R.-M. Chen, "Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem," Expert Systems with Applications, vol. 38, no. 6, pp. 7102-7111, June 2011, doi: 10.1016/j. eswa.2010.12.059.

[12]. H. Dai, W. Cheng and P. Guo, "An Improved Tabu Search for Multi-skill Resource-Constrained Project Scheduling Problems Under Step-Deterioration," Arabian Journal for Science and Engineering, vol. 43, no. 6, pp. 3279-3290, Jan. 2018, doi: 10.1007/s13369-017-3047-4.

[13]. H. Saad, R. Chakrabortty, S. Elsayed and M. Ryan, "Quantum-Inspired Genetic Algorithm for ResourceConstrained Project-Scheduling," IEEE Access, vol. 9, pp. 38488-38502, 2021, doi: 10.1109/ACCESS.2021.3062790.

[14]. H. F. Rahman, R. K. Chakrabortty, and M. J. Ryan, "Memetic algorithm for solving resource constrained project scheduling problems," Automation in Construction, vol. 111, March 2020, doi: 10.1016/j.autcon.2019.103052.

[15]. K. M. Sallam, R. K. Chakrabortty, and M. J. Ryan, "A two-stage multi-operator differential evolution algorithm for solving Resource Constrained Project Scheduling problems," Future Generation Computer Systems, vol. 108, pp. 432-444, July 2020, doi: 10.1016/j.future.2020.02.074.

[16]. R. V. Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," International Journal of Industrial Engineering Computations, vol. 7, no. 1, pp. 19-34, 2016, doi: 10.5267/j. ijiec.2015.8.004.

[17]. K. Mihály, G. Kulcsár: A New Many-Objective Hybrid Method to Solve Scheduling Problems https://ijiemjournal.uns.ac.rs/images/journal/volume14/IJIEM_342.pdf

[18]. Tóth, T.: Termelési rendszerek és folyamatok, Miskolci Egyetemi Kiadó, Miskolc, 2004.

[19]. Tóth, T.: Tervezési elvek, modellek és módszerek a számítógéppel integrált gyártásban. Miskolci Egyetemi Kiadó, 2006.

Adathordozó melléklet tartalma

/Programkód

/Klines

/FrontEnd

/Szerver

CppREST.a

CppCrow.cpp

/Szakdolgozat

Szakdolgozat.pdf

Feladatkiírás.pdf

/Bemeneti fájl

j301_1.sm