AURORA Órarendszerkesztő program

Szemléltető leírás (a 0.6 verzió alapján)

Horváth Dávid

2016

Tartalomjegyzék

1. Bevezetés

A számítógépek elterjedése előtt az órarendet kézzel kellett elkészíteni, melynek eszköze általában egy tábla és az arra feltűzdelhető – a tanórákat jelképező – papírcetlik voltak. A táblára előre fel volt rajzolva az időbeosztás, és általában a tanárok – esetleg az osztályok – névsora. Sok helyen még most is ezzel a módszerrel zajlik a tanórák órarendbe szervezése. Az így készített órarendnek lelke van, a készítő feltehetően jól ismeri az iskola felépítését, a tanárok egyéni igényeit, így egészen komplex feltételrendszert tud érvényesíteni.

A kézi felhelyezésnek megvan az az előnye, hogy bármilyen különleges vagy előre nem látott kérés ad-hoc módon teljesíthető. Ezt számítógép segítségével nehéz utánozni, mivel a programok előre rögzítettek – még a legrugalmasabban tesreszabható, konfigurálható algoritmusnak is megtalálhatók a gyenge oldalai.

Azonban kétségtelen tény, hogy a számítógépek egyre gyorsabbak, és alaptermészetüknél fogva fáradhatatlanul végrehajtják a rájuk bízott feladatot. Erre építve pedig a programozó dolga, hogy a program a kézimunka minőségét megközelítse, illetve bizonyos előre meghatározható feltételek teljesítésében felül is múlja. Erre vállalkoztunk az AURORA Órarendszerkesztő fejlesztésével.

Az órarend számítógépes elkészítésének problémáját három részre bonthatjuk:

1. Az ideális adatmodell megtalálása – azaz: hogyan képezzük le számítógép által értelmezhető formában az órarend elemeit – akár kifejezetten a kézi készítésnél használt segédeszközök imitálásával. Az adatmodellnek elég rugalmasnak kell lennie ahhoz, hogy bármilyen szituációt képes legyen leírni.
2. Felhasználóbarát reprezentáció, melynek segítségével a program használója hatékonyan tudja felvinni és átlátni az adatokat.
3. Hatékony algoritmus az órarend automatikus generálásához. Természetesen az órarend kézi felvitelére illetve módosítására is lehetőséget kell adni.

Előre leszögezzük, hogy jelen állapotában az AURORA – szerintünk – az első pontban a legerősebb, mivel a főcél kezdettől fogva a lehető legnagyobb rugalmasság és bővíthetőség volt. Ez persze nem jelenti azt, hogy a felhasználói felület ne lenne jól használható, vagy hogy a program ne rendelkezne hatékony generáló algoritmussal.

A felhasználói felület tervezése és fejlesztése folyamatosan zajlik. A felhasználói felület használatáról részletesen a Felhasználói kézikönyvben olvashat!

A beépített generáló algoritmusról lentebb még részletesebben szólunk. Ezen felül a program támogatja egyedi generáló algoritmusok futtatását is. Az egyedi algoritmusok készítését megegyezés alapján vállaljuk. Lehetséges, hogy a későbbiekben ehhez részletesen dokumentált fejlesztői környezetet adunk ki, melynek segítségével (a kikötött feltételek betartása mellett) harmadik felek is fejleszthetnek egyedi algoritmusokat.

2. Az adatmodell

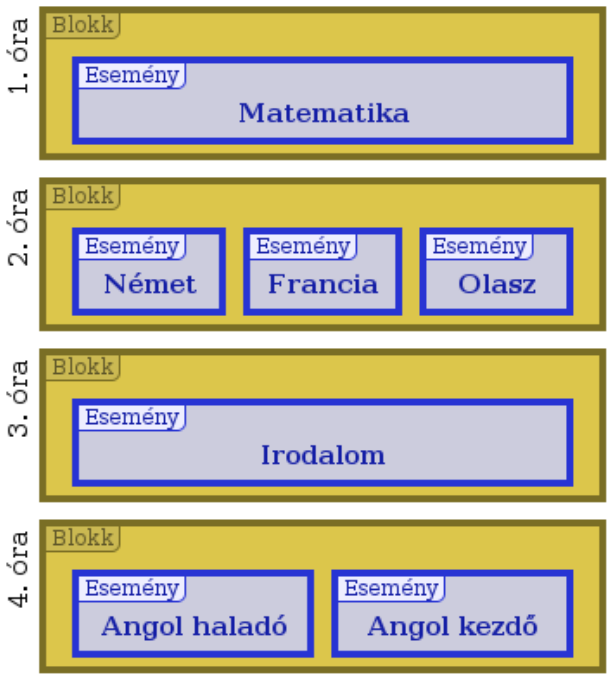
2.1. Blokkok és események

A legtöbb nyilvántartás-alapú programban elkülöníthető valamilyen főobjektum, mely köré az összes többi objektum és a program legtöbb funkciója épül. Egy webáruházban ez lehet a „termék”, egy hírportálon a „hír”, egy autókölcsönzőnél pedig az „autó”.

Ha az ARORA Órarendszerkesztő programban kellene megnevezni egy ilyen főobjektumot, akkor az mindenképpen a „blokk”. Bár a blokk fogalma a program egyik alapköve, lényegét nem is olyan könnyű összefoglalni, viszonylag összetett objektumról van szó.

Kiindulásképpen leegyszerűsíthetjük úgy, hogy a blokk nem más, mint egy tanóra. Időhosszal rendelkezik, és felhelyezhetjük egy tábla tetszőleges időpontjára.

Egy-egy órarend-dokumentumban korlátlan számú tábla hozható létre. Egy blokk egy táblán egyszerre természetesen csak egy helyen szerepelhet, de több táblára is fölhelyezhetjük. Így megoldható például, hogy az órarendből több verziót is készítsünk, és később döntsünk közöttük.



1. ábra: Blokkok és események

De miért „blokk”? Valójában egyetlen blokkban több esemény (tanóra) is elhelyezhető. Ezeket az eseményeket a blokk összefogja, így garantálható, hogy az órarendben azonos időpontra kerülnek.

Tehát az egyszerű esetben ugyan egy-egy blokkban valóban csak egyetlen esemény található, de a blokk jelentheti események fürtjét is.

Ezt sokmindenre fel lehet használni. Tipikus eset, amikor az osztály két fele két külön órán vesz részt, például nyelvi bontás miatt. Ekkor célszerű két külön eseményként regisztrálni a bontott órát, de egy blokkba helyezni. A bal oldalon ilyen helyzeteket igyekeztünk szemléltetni.

Egyetlen blokkban tetszőleges számú esemény lehet. Az események között nem kell kapcsolatnak lennie, akár teljesen független (például különböző osztályokhoz tartozó) tanórák is kerülhetnek egy blokkba.

Egy-egy eseményhez több (tetszőleges számú) tanár, osztály stb. kerülhet. Az összevont órák kezelése így meglehetősen egyszerű. A tanárok, osztályok stb. kezelését később tárgyaljuk.

> Terveink szerint későbbi fejlesztések során a blokk lehetőségeit bővíteni fogjuk. Ha ez megvalósul, akkor lehetőség lesz egymáshoz képest konkrét időtartammal vagy dinamikusan eltolt események elhelyezésére a blokkban. Így a dupulaórák kényszerítése is egyszerű lenne. Ez azonban nehézségeket okoz a generáló algoritmusban, így egyelőre elvetettük a megvalósítását.

2.2. Ciklusok

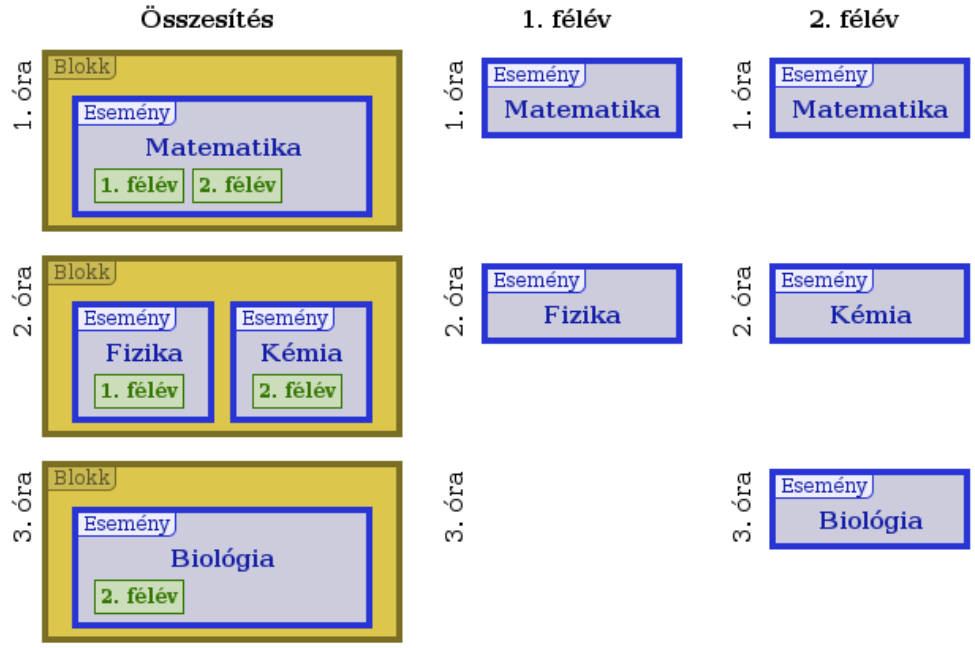
A legtöbb órarend nem egyetlen állandó hétből áll, gyakori az A-B hetes vagy kétféléves szétbontás, vagy akár a kettő egyszerre. Ezekkel a váltóhetekkel szemben általában elvárás, hogy órarendjük minél kevesebb eltérést tartalmazzon. Így például az első és második félévben váltakozhat két

tantárgy, melyeknek heti óraszáma a teljes évre számítva csak 0,5; a többi tantárgy órái pedig megmaradnak a helyükön. Az ilyen egymást váltó órarendi „idősíkokat” ciklusnak nevezzük.

> Szándékosan nem a „hét” elnevezést választottuk. Ha például olyan kéthetes órarendet szeretnénk, ahol a két hét tetszőlegesen eltérő lehet, akkor a két hetet egyetlen ciklusban célszerű felvenni. Ehhez szükséges a megfelelő időbeosztás létrehozása. Az időbeosztásokról lentebb szólunk.

A blokkokban elhelyezett eseményeket tetszőlegesen egy vagy több ciklushoz rendelhetjük. Tehát nem a blokkot rendeljük a ciklushoz, hanem az eseményt. Így megvalósítható, hogy egy blokkban olyan eseményeket helyezzünk el, melyek eltérő ciklushoz vagy ciklusokhoz vannak rendelve. Ha például egy blokkban ugyanannak az osztálynak két eseménye is szerepel, az alapesetben ellentmondás lenne, de ha nincs közös ciklusuk (például az egyik az A-héthez, a másik pedig a B-héthez lett rendelve), akkor nyilvánvalóan megengedhető.

Más-más ciklus szerint nézve tehát más-más események látszódhatnak az órarendből:



2. ábra: Blokkok és nézeteik ciklusonként

Egy órarend-dokumentumban tetszőleges számú ciklus lehet, így (többek között) az alábbi ritkább szituációk is kezelhetők:

- Két félév és azokon belül A-B hét
- A-B-C-D stb. hét

De a ciklusok akár más célokra is felhasználhatók, például nem órarendi események, tevékenységek számontartásához, melyek nem zárják ki a tanítást (pl. készenléti idők).

> Egyedi generáló algoritmus esetén megvalósítható, hogy a blokkok közötti ütközések és a ráépülő megkötésrendszert cilkusonként eltérően, az igényeknek megfelelően kerüljön figyelembevételre.

2.3. Erőforrások és ütközések

Két tanóra nyilvánvalóan nem zajlódhat egyszerre, ha ugyanaz a terem vagy tanár van kijelölve a számukra, vagy ha például ugyanaz a teljes osztály vesz rajtuk részt. Az események tehát bizonyos jellemzőik miatt összeférhetetlenek lesznek, és ilyenkor nem történhetnek egy időben.

Amikor két esemény nem történhet egy időben (például azonos tanár miatt), akkor a két esemény közötti ütközésről beszélünk. Egy blokk ellentmodásos, ha ütköző események vannak benne, (kivéve, ha nincs közös ciklusuk). Illetve két blokk ütközik, ha van egy-egy eseményük, amelyek rendelkeznek közös ciklussal és ütköznek.

> A program jelenleg megengedi ellentmondásos blokkok létrehozását, mivel a blokkok közötti ütközés definícióját – ami az automatikus generálás alapját képezi – nem befolyásolja. Az ilyen blokkok létrehozása azonban csak extrém esetben ajánlott (például nem véglegesített bontott óráknál).

Két esemény között akkor jöhet létre ütközés, ha azonos erőforrást használnak. Az AURORA Órarendszerkesztőben a tanár, osztály, terem, eszköz stb. fogalmak mind „erőforrások”.

Vannak más, eseményekhez rendelhető jellemzők, amelyek nem generálnak ütközést, ezeket címkéknek nevezzük. Ilyen például a „tantárgy”, elvégre két esemény még nem ütközik pusztán azért, mert mindkettő matematika óra.

Tehát a programban a blokkok közötti összeférhetetlenség a hozzárendelt erőforrások alapján kerül megállapításra.

> Bizonyos esetben lehetséges, hogy azonos erőforrást használó események nem ütköznek. Ez akkor fordulhat elő, ha az adott erőforrás darabszáma 1-nél nagyobb, vagy ha az erőforrás bontható, és a két esemény az erőforrás más részeit használja. A darabszámokról és bontásokról lentebb még részletesen szólunk.

Létrehozhatunk akár olyan erőforrást is, ami nem köthető szorosan valamely könnyen megnevezhető objektumhoz (a programban ez jelenleg az „Egyéb erőforrás” beállítással valósítható meg). Ezek az elvont jellegű erőforrások ugyanúgy ütközést generálnak az események között, mint a normál erőforrások (tanár, osztály stb.), így segítségükkel minden mellékhatás nélkül kikényszeríthetjük, hogy két blokk ne kerülhessen azonos időpontra.

Egy-egy eseményhez tetszőleges számú erőforrás illetve címke rendelhető. Létrehozhatunk összevont órákat is, ha egy tanórához több osztályt is hozzáadunk. Ugyanígy nincs akadálya, hogy egy eseményhez rendeljünk több tanárt, de akár több tantermet, tantárgyat (stb.) is.

A programban az erőforrásokhoz és címkékhez színt is rendelhetünk. Ez átláthatóbbá teszi az összesített nézetet.

2.4. Erőforrások felbontása, csoportok

Szinte minden órarendben előfordul olyan eset, amikor egy osztály több csoportra bomlik, és az egyes csoportok különböző foglalkozáson vesznek részt. Ráadásul egy-egy osztálynak többféleképpen felbonthatónak kell lennie, mivel más-más tantárgy esetében más-más lehet a felbontás (például matematikából illetve idegen nyelvből). Ennek lehetővé tétele érdekében minden erőforráshoz tetszőleges számú csoportbontás definiálható, melyen belül tetszőleges számú csoport adható meg.

A csoportbontásokra az alábbi szabályok érvényesek:

1. Egy adott csoportbontás csoportjai összeadva a teljes osztályt (erőforrást) adják ki (ha mégis van kimaradó rész, akkor azt is hozzá kell adni csoportként)
2. Egy csoportbontáson belül két különböző csoportnak nincs metszete (ha mégis van metszet, akkor azt is külön csoportként kell felvinni – bár ez esetben valószínűleg inkább több csoportbontás lenne logikus)
3. Két különböző csoportbontás egy-egy csoportja mindig ütközik

Jelenleg nincs lehetőség konkrét tanulók kezelésére illetve azok csoporthoz rendelésére, de a későbbiekben valószínűleg ennek megvalósítására is sor kerítünk. Az ilyen, hozzárendelt tanulókkal definiált csoportok esetében a fenti állítások – értelemszerűen – nem feltétlenül lesznek igazak.

Ezen a ponton már kimondhatjuk, hogy egy-egy eseményhez valójában nem erőforrásokat rendelünk, hanem az erőforrások valamilyen részhalmazát. Ez lehet:

1. A teljes erőforrás (alapértelmezett)
2. Az erőforrás egy csoportja
3. Az erőforrás csoportjaiból halmazműveletek (unió, metszet, negáció stb.) útján képzett összetett részhalmaz (általános eset)

Már említettük, hogy összevont órákat képezhetünk, ha egy eseményhez több osztályt is rendelünk. Mostmár láthatjuk, hogy összevont csoportokat is használhatunk, ha egy eseményhez különböző osztályok csoportjait rendeljük (például amikor a nyelvi csoportok több osztályból vannak képezve).

2.5. Időbeosztások

A legtöbb órarendgeneráló program egy egyszerű táblázatba helyezi el a tanórákat. A táblázat oszlopai a napokat, a sorok pedig az órasorszámokat jelentik. Ez a megközelítés egy jól átlátható táblázatot eredményez, ami egy generáló algoritmus számára is könnyen kezelhető struktúrát biztosít. A szabad kapacitásra vonatkozó beállítások is könnyen megfogalmazhatók (pl. ez és ez a tanár ebben és ebben a cellában nem ér rá).

Azonban ez a megoldás magában hordoz néhány súlyos kötöttséget is, bár ez a legtöbb iskolában ez nem jelent gondot. Mi azonban a nagyobb rugalmasság érdekében olyan megoldást kerestünk, ami minden lehetséges szituációt képes hatékonyan kezelni, annál is inkább, mivel már a program első használatakor is szükség volt a kötött táblázatos formából való kilépésre.

Előfordul például, hogy az alsósok hamarabb megebédelnek, és ebéd után még vannak óráik. Ekkor az ebéd utáni órák később kezdődnek. Ugyanígy előfordulhat, hogy egy osztálynak külső helyszínen van foglalkozása, ahonnan a csengetési rendhez képest késve érnek vissza.

Megjegyezzük, hogy egyes programok támogatják az elcsúsztatás ilyen egyszerű eseteit. A táblázatos megoldással járó alapvető kötöttség azonban megakadályozza a tetszőleges bonyolultságú szituációk kezelését.

A mi programunk tehát nem táblázatcellákkal dolgozik, hanem egy másodperces pontosságú idővonallal. Ez azt jelenti, hogy egy-egy blokk a táblán tetszőleges időintervallumon helyezkedhet el, ehhez a megfelelő kezdőidőpontra kell felrakni, és a megfelelő hosszúságúra beállítani. Így lehetővé válik például, hogy két osztály némileg elcsúsztatott órakezdési időpontokkal rendelkezzen.

Az automatikus generálás számára tehát valahogyan pótolni kell a táblázatcellákat, mivel a programnak tudnia kell, milyen időpontokra szabad felhelyeznie az egyes blokkokat. Ezt pedig úgy kell lehetővé tenni, hogy az alábbiak egyszerre teljesüljenek:

1. Tetszőleges saját időbesztással rendelkezhessen bármely blokk
2. Osztályonként (erőforrásonként) tömegesen beállítható legyen az időbeosztás
3. Több osztályhoz lehessen ugyanolyan időbeosztást hozzáadni és egyszerre módosítani
4. Lehessen generálni egy alapértelmezett időbeosztást (pótolva azt a helyzetet, hogy a táblázatos megoldásnál maga a táblázat eleve adott, és nem kell különféle időbeosztásokkal

foglalkozni)

Tehát eljutottunk az „időbeosztás” fogalmához a korábbi „táblázat” helyett. A mi időbeosztásunk és táblánk együttesen megfelel a más programokban látható táblázathoz, azonban sokkal rugalmasabb.

Azt mondtuk, minden blokknak saját időbeosztása lehet, ez azonban csak némi áttételellel igaz. Alapesetben az osztályokhoz van értelme időbeosztást rendelni, így úgy döntöttünk, hogy az időbeosztásokat az erőforrásnál (és címkénél) kell megadni, nem a blokknál. A blokk időbeosztása pedig a hozzárendelt erőforrás (illetve címke) időbeosztását veszi fel. Így még mindig lehetőség van arra, hogy egyesével adjuk hozzá az időbeosztást, mégpedig külön időbeosztást definiáló „egyéb” erőforrások hozzárendelésével, ennek módját lásd a lentebbi példában.

Tehát az időbeosztások kezelése három lépcsős.

Első lépcső: a dokumentumban tetszőleges számú időbeosztás vehető fel, ezek később is módosíthatók, törölhetők, és a listát természetesen bármikor bővíthetjük. Ilyen értelemben az időbeosztás ugyanolyan független objektum, mint a ciklus, az erőforrás, a tábla, a blokk stb.

Második lépcső: minden erőforrásnak megadhatjuk, hogy mely időbeosztás tartozik hozzá. Természetesen ez lehet ugyanaz akár az összes osztálynál. Ez ciklusonként felülírható. Tehát például egy osztály időbeosztása félévenként eltérhet.

Harmadik lépcső: az erőforrásokat a blokk eseményéhez rendeljük, és a blokk felveszi az adott esemény időbeosztását. A felvett órarend ciklusonként eltérhet, ha az események más ciklushoz lettek rendelve – vagy ha az erőforrásnál ciklusonként különböző időbeosztás lett hozzárendelve. Ha egy blokkban (ciklusonként) nincs olyan erőforrás, ami időbeosztást adna meg (vagy üres időbeosztással rendelkezik), akkor az automatikus generálás sikertelen lesz. Ha a blokkban (valamely ciklusban) több időbeosztással rendelkező erőforrás van, akkor a blokk időbeosztásában a közös időpontok lesznek megtartva; ha nincs közös időpont, a generálás szintén sikertelen lesz. A hiányzó vagy rosszul beállított időbeosztások a programban ellenőrizhetők.

Most vegyük sorra a keletkezett plusz előnyöket:

1. Elcsúsztatott órarendek hozhatók létre
2. Bármilyen eltérés lehet, így például beállítható, hogy akár egyetlen oszálynak (vagy csoportnak) legyen szombati foglalkozása
3. Az időbeosztás tetszőleges erőforráshoz rendelhető (nem csak osztályokhoz), így egyéb eseményeket (nem tanórákat) is belevehetünk az automatikus generálásba, amelyekkel az ütközésmentesítés megtörténik
4. Kézileg az időbeosztáson kívüli helyre is feltehetjük a blokkot. A maradék blokkokat a generáló algoritmus az időbeosztás betartásával helyezi el

A 3. pontra egy példa: egy tanárnak 2 alkalommal kötelező délelőtti elfoglaltsága van, melyek összesen 10 adott időpont valamelyikén kezdődhetnek. A generáló algoritmus ekkor kiválaszt a megadott időpontok közül kettőt, amelyek nem ütköznek a tanár óráival. Ez így oldható meg a programban:

1. Létrehozunk egy új időbeosztást a tíz lehetséges kezdőidőponttal
2. Létrehozunk egy új „egyéb” erőforrást (vagy címkét), melyhez hozzárendeljük ezt az időbeosztást
3. Létrehozunk egy új blokkot egy eseménnyel (ez lesz maga az egyik elfoglaltság), hozzárendeljük a tanárt, és a 2. pontban létrehozott erőforrást
4. Duplikáljuk a blokkot, hogy 2 db legyen belőle

Fontos, hogy ilyenkor „egyéb” erőforrást használjunk, és ne egy osztályt hozzunk létre (mondjuk a

„következetesség” kedvéért). A generáló algoritmusban ugyanis az osztályokra számtalan plusz megkötés vonatkozik, az „egyéb” erőforrás azonban (az ütköztetést leszámítva) teljesen semleges. A fenti esetben mindegy, hogy erőforrást vagy címkét használunk, mivel a tanár (mint erőforrás) eleve ütközteti a blokkokat.

A generáló algoritmus egy-egy blokkot a számított időbeosztása szerint megmaradt helyek valamelyikére fog elhelyezni. A blokkra vonatkozó időbeosztás a következőképpen kerül kiszámításra:

1. Ciklusonként kiszámítja az időbeosztást a ciklushoz rendelt események erőforrásainak időbeosztásainak metszeteként
2. Ciklusonként hasonlóan kiszámítja az időkorlátot (az időkorlátokkal a következő fejezetben foglalkozunk), és az időbeosztásokra érvényesíti (mindegyikre a vele azonos ciklusban számolt időkorlátot)
3. A ciklusonként (az időkorlátok figyelembevételével) számított időkorlátoknak a metszetét veszi

Ha így nem marad időpont, akkor a blokkot nem lehet felhelyezni, és a generálás sikertelen lesz. A programban az ilyen esetek is ellenőrizhetők.

2.6. Időkorlátok

Alapvető követelménynek tekinthetjük, hogy a tanárok és tantermek számára foglaltságot lehessen beállítani. A legjellemzőbb eset, ha egy tanár más iskolában is tanít, és a megegyezés alapján valamely napokat vagy idősávokat a másik iskola részére tartunk fenn. Ennek érvényesíthetősége érdekében a programban bármely erőforráshoz és címkéhez időkorlátok adhatók meg, melyek az időbeosztáshoz hasonlóan ciklusonként felülírhatók.

Fontos különbség az időbeosztáshoz képest, hogy az időkorlát nem képez önálló objektumot, mindig csak az adott erőforrás illetve címke belső adata.

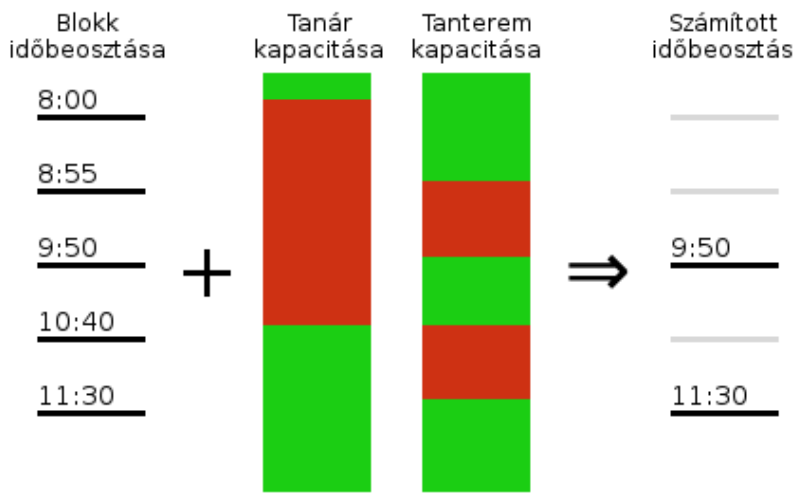
Jelenleg nincs lehetőség gyenge feltételek megadására (pl. „jó lenne, ha ekkor nem lenne órája, de megengedhető, hogy legyen”). A program magja a gyenge feltételeket jelenleg is kezelni tudja, de az alapértelmezett generáló algoritmus ezt nem használja ki, illetve egyelőre a felhasználói felület sem támogatja. Egyedi algoritmussal ez a gyengeség pótolható.

Az időbeosztások gyors számolása végett az időkorlátok mindig a kezdőidőpontra vonatkoznak, így ha egy blokk vége egy adott pozíción „kilóg” a szabad sávból, a pozíció akkor is megengedett.

Később tervezzük az időbeosztások számításának átalakítását – úgy, hogy többféle viselkedésű időkorlát felvitelére lesz lehetőség. Ugyancsak tervezzük, hogy a gyenge feltételekre komplex megoldást készítünk, amely a felületen is könnyen adminisztrálható lesz.

Az időkorlátok szükségessége leginkább a tanároknál és a tantermeknél merül fel, más esetekben is hasznos lehet. Például beállíthatjuk, hogy adott tantárgy ne legyen a reggeli vagy éppen a későbbi órákban. A nehéz tantárgyak megállapítása és kezelési stratégiája persze meglehetősen szubjektív (akár csak a nehéz napoké, ahogy arra lentebb még visszatérünk).

A fentiekben már utaltunk rá, hogy a blokk (ciklusonkénti) időbeosztásának számítása úgy történik, hogy az erőforrásokból és címkékből származó időbeosztások és időlimitek metszetét vesszük. Az alábbi ábrán egy egyszerű példát láthatunk erre a mechanizmusra:



3. ábra: Időkorlátok hatása az időbeosztásra

Ha definiálni akarjuk, az időkorlát egy kételemű adatstruktúrának tekinthető. Az első elem egy logikai adat, mely tárolja, hogy az időkorlát szabad státuszból indul-e (matematikai értelemben: az időben visszafelé mutató nyílt intervallumhoz szabad vagy foglalt státuszt rendelünk), A második elem egy újabb halmaz, mely időpontokat tartalmaz. Minden időpont átváltja a státuszt az ellenkezőre. Így az alábbi struktúra:

{ szabad; { Hé 10:00; Hé 15:00; Ke 0:00; Sze 0:00 } }

azt jelenti, hogy az erőforrás hétfőn 10 és 15 óra között illetve kedden egész nap foglalt, egyébként szabad.

3. A generáló algoritmus

3.1. Algoritmusok

Az AURORA Órarendszerkesztő program tetszőleges számú generáló algoritmus használatt teszi lehetővé. Az alapcsomagban három algoritmus található, ezek közül kettő egyszerű ütközésmentesítéshez használható. Iskolai órarendek készítésére a harmadik, „Iskolai” elnevezésű algorimus lett tervezve.

> Ahogy már említettük, ez egyedi algoritmusokkal bővíthető. Az egyedi algoritmusokat jelenleg kívülről nem lehet a programhoz adni, így ezt csak mi tudjuk megtenni.
>
> Úgy tervezzük, hogy egy egyszerű fejlesztői környezet is elérhető lesz a programhoz, mellyel az egyedi algorimusok tesztelhetők. Szintén napirenden van a plugin-rendszer fejlesztése, melyen keresztül az egyedi algoritmusok (és más külön fejlesztett funkcionalitások) külső JAR-fájlként lesznek betölthetők a programba.

A továbbiakban az „Iskolai” algoritmussal foglalkozunk.

3.2. Működési elv

Az órarendgenerálás meglehetősen összetett feladat, matematikai értelemben is a „nehéz” problémákhoz tartozik. A matematika jelenlegi állása szerint a hasonló problémákra nem létezik olyan egyszerű megoldási módszer, mely belátható időn belül garantáltan szolgáltatja a legjobb megoldást. Így tehát valamilyen trükkre vagy egyszerűsítésre van szükség.

Két (alapvetően különböző) módszerrel állhatunk neki a megoldásnak:

1. Nem-teljes lokális keresés: optimalizáló algoritmus, mely egyre jobb megoldásokat generál. Futási ideje skálázható, a megoldás minősége (jó esetben) ezzel arányos lesz. Azonban nem garantálható, hogy (akár tetszőleges idő alatt) eljut a legjobb megoldáshoz (vagy akár egy használható megoldáshoz). Gyakran használják olyan nehéz problémák megoldásánál, ahol közelítő megoldás is elfogadható.
2. Teljes metódus: olyan algoritmus mely vagy megtalálja a (legjobb) megoldást, vagy bizonyítja a megoldhatatlanságot (és ezt egy előre behatárolható – bár akár exponenciális arányú – időn belül megteszi). Általában erősebben épül matematikai módszerekre, mint a nem teljes metódusok (például a lokális keresés). Előnye, hogy biztos választ ad. Hátránya, hogy nehéz problémák esetén akár exponenciális arányú ideig futhat; és menet közben megállítva a legtöbb esetben nem nyerhető ki belőle használható megoldás (nincs tárolva egy „mindenkori legjobb megoldás”, mint a lokális keresésnél).

A legtöbb meglévő órarend-program az első módszert használja. Lokális kereső algoritmusok igen régen használatban vannak, és a kifinomult módszerekkel már régóta elfogadható hatékonysággal lehet az órarendhez hasonló problémákat kezelni.

Mi ezzel szemben a teljes algoritmus mentén próbálkoztunk, és az általunk használt módszer elsősorban adatelemzésre és logikai programozásra épít.

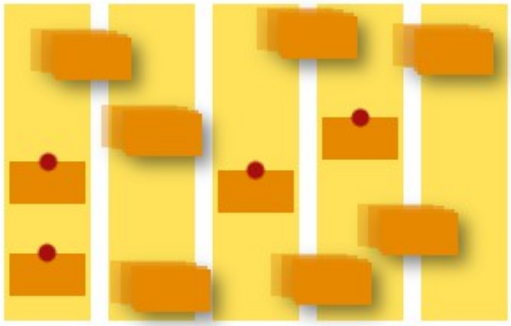
> A logikai programozás – a lokális keresési módszerekhez hasonlóan – szintén évtizedekre nyúlik vissza. Egyik fő eleme viszont, konkrétan a logikai formulák megoldhatóságának eldöntése (SAT), igen kemény matematikai probléma, amelynek kezelésére évtizedekig nem volt hatékony eljárás. A közelmúltban azonban jelentős áttörések történtek ezen a téren.
>
> A legújabban kifejlesztett (teljes) SAT-megoldó algoritmusok már annyira hatékonyak, hogy igen nagy feltételrendszerekkel is megbírkóznak (csak rendkívüli esetben burjánzanak el az exponenciális futási

ágak). Tehát a teljes típusú SAT-megoldás ma már felveheti a versenyt a lokális algoritmusokkal.

A SAT-probléma optimalizációs változatára (melyben gyenge feltételek kezelésére is lehetőség van) kifejlesztett algoritmusok szintén rendkívüli fejlődésen mentek keresztül az utóbbi években. A programunk számára ez lehetővé teszi másodlatos időlimitek és egyéb gyenge feltételek használatát (egyelőre egyedi algoritmussal).

Az algoritmus a legtöbb feltételrendszert ciklusonként érvényesíti, de ezt külön nem említjük meg a továbbiakban.

3.3. Fix órák kezelése



4. ábra: Fix órák

Ha egy táblán a generálás indításakor már vannak felhelyezett blokkok, akkor a generáló algoritmus azokat a helyükön hagyja, így valósul meg a fixen előre felrakott blokkok kezelése.

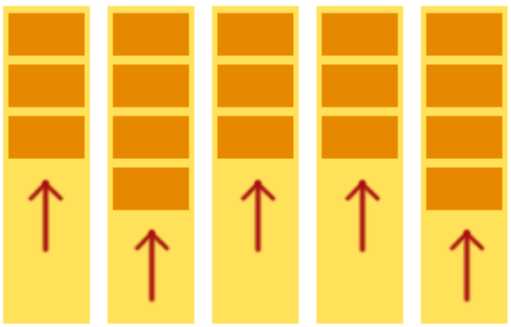
Javasolt a fix blokkokat egy külön erre a célra fenntartott táblán elhelyezni, és generálás előtt ezt a táblát duplikálni (másolni). Így megmaradnak a fix blokkok, ami információként és újabb generálás kiindulási pontjaként is fontos lehet.

Az algoritmus már felhelyezett blokkokkal teljesen „elnéző”. Lehetnek időbeosztáson vagy akár időlimiten kívül, ez nem fogja zavarni a generálást; természetesen az újonnan felrakásra kerülő blokkokra alkalmazásra kerülnek ezek a feltételek is.

Generálás előtt egyes blokkok már szerepelhetnek a táblán kézi felrakás vagy egy korábbi generálás maradványaként. Lehetséges csak bizonyos blokkokat kiválasztani automatikus felhelyezésre. Azonban az „Iskolai” algoritmusnál az így megoldott többlépcsős felrakás nem igazán hatékony, mivel már az első néhány blokkra alkalmazza a teljes feltételrendszert (pl. a gravitációt).

A későbbiekben olyan algoritmusokat is biztosítunk majd, melyeknek célja nem a teljes blokkhalmaz felhelyezése, hanem kisszámú blokk kevesebb feltétel betartása melletti felrakása, kifejezetten a „nagy” generálás előtti előkészítési fázisra.

3.4. Gravitáció a reggel irányába



5. ábra: Gravitáció

Az osztályonkénti órarenddel szemben elvárás, hogy reggel kezdődjön, ne legyenek benne lyukasórák, és a napi óraszám is kiegyensúlyozott legyen. Az algoritmus a reggel irányába tolja az órákat, és amennyire lehet, igyekszik legfeljebb 1-gyel eltérő napi óraszámokat kikényszeríteni.

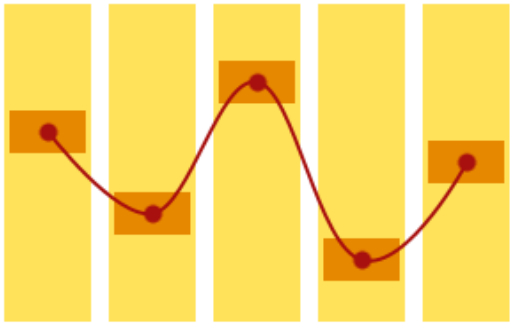
Fix órák behelyezésével ez a működés befolyásolható. Ha például kézzel helyezünk el órákat késői időpontokra, azok a helyükön maradnak, és a maradék blokkok kerülnek gravitálásra.

A gravitációban érintett (vagy azokat metsző) kezdőidőpontokra helyezett blokkok átugrásra kerülnek, de a napi óraszám megállapításában természetesen részt vesznek.

3.5. Osztályonkénti tantárgy-elosztás

Osztályonként a program az alábbi lehetséges elosztásokat végzi az azonos tantárgyú blokkokkal:

1. Ha a blokkok száma a napok számánál kevesebb, akkor két blokk nem lehet egy napon
2. Ha a blokkok száma a napok számával egyenlő, akkor minden napra egy-egy blokk kerül
3. Ha a blokkok száma nagyobb a napokénál, akkor az előző szabályok keverednek, és egymás utáni elhelyezések (duplaórák) lesznek kényszerítve



6. ábra: Tantárgy-elosztás

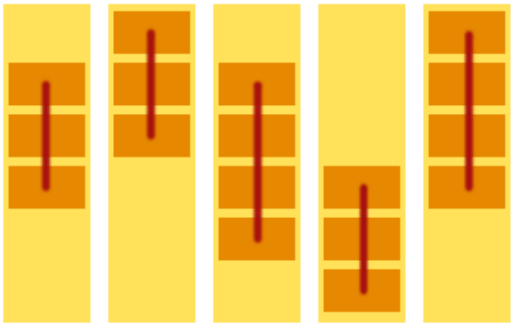
Valójában ennél bonyolultabb a helyzet. Az azonos órák blokkjai nem egyformák, akkor nem csak a feltételek optimalizálása válik nehezebbé, hanem az elosztási variációk is csökkenhetnek.

Kevesebb blokk esetén további feltételek is érvényesülnek. Amikor például 5 napra 2 blokk jut, akkor azok nem kerülhetnek szomszédos napra.

3.6. Tanárok lyukasóráinak csökkentése

Sokszor egy tanár számára nagyon előnytelen, ha túl sok lyukasórája van. Ezért a generálás beállításainál kiválaszthatók azok a tanárok, akiknek:

1. Ne engedjen dupla lyukasórát (75 percnél távolabbi kezdőidőpontok)
2. Egyáltalán ne engedjen lyukasórát (25 percnél távolabbi kezdőidőpontok)



7. ábra: Lyukasóra-csökkentés

Mivel az időbeosztás tetszőleges lehet, a lyukasórákat a kezdőidőpontok közötti távolság alapján definiálja a program. Lyukasóra van két azonos napon felhelyezett szomszédos blokk között, ha kezdőidőpontjaik távolsága 25 percnél nagyobb, illetve dupla lyukasóra, ha a távolság 75 percnél nagyobb.

Jelenleg ezek a beállítások a felhasználó által nem módosíthatók.

A lyukasórák kezelése egyedi algoritmussal természetesen tovább finomítható.

3.7. További feltételrendszerek

A generálás speciális beállításainál a lyukasóra-mentesített tanárokon kívül még két lista adható meg.

Az egyikben azon tanárok listája állítható össze, akiknek minden nap legalább két órájuk kell legyen (vagy pedig egy sem, tehát pontosan 1 nem lehet). Ennek elsődleges értelme a vidékről bejáró tanárok óráinak sűrítése: ne kelljen egyetlen tanóra miatt beutazniuk.

A másiknál címkék adhatók meg, amelyek a hozzárendelt blokkokat tanáronként szétosztják külön napokra (a tantárgyi elosztáshoz hasonlóan). Ennek egyik felhasználási területe, hogy egy-egy osztálynak a saját osztályfőnökével mindig legyen órája. Nem szükséges ilyen beállítás, ahol ezt önmagában egy tantárgy is biztosítja (például az alsós osztályokban a magyar órák).

Egyéb különleges feltételek érvényesítésére egyedi algoritmusokkal van lehetőség.

4. A program főbb hiányosságai

Furcsának tűnhet, hogy erre is kitérünk; azonban ebben a dokumentumban a tájékoztatás az elsődleges célunk. Az AURORA Órarendszerkesztő jelenleg még kezdeti stádiumban van, így természetes, hogy megvannak a maga hiányosságai. Reméljük azonban, hogy a program által nyújtott egyéb előnyök ezt már most is ellensúlyozzák.

A lentebb felsorolt hiányosságok nagyrészt olyan szolgáltatások, amelyek valamely más órarendszerkesztő programban megtalálhatók. Ugyanakkor éppen azért soroljuk fel őket, hogy jelezzük, későbbi fejlesztésként lehet számítani ezek megjelenésére az AURORA programban.

4.1. Gyenge feltételek

Mint már írtuk, a program magja valójában támogatja a gyenge feltételeket, és ez egyedi algoritmussal ki is használható. A beépített algoritmus(ok)ban a gyenge feltételek hiánya nem tűnik drasztikusnak, mivel a jelenlegi algoritmus eléggé hatékonyan megtalálja a jó megoldást, ha létezik. Ha pedig nem létezik, azt külön jelzi, és a felhasználónak lehetősége van rá, hogy saját belátása szerint módosítson az adatokon. Ebben az blokk-ellenőrzési lehetőség is segít.

4.2. „Nehéz napok”

Megoszlanak a vélemények a „nehéz” napokat illetően. Minden diák számára más a nehéz, és van, aki kifejezetten szereti, ha a hét nehéz és könnyű napokra oszlik.

Mindenesetre sokszor felmerül az igény, hogy be lehessen állítani „nehéz” tantárgyakat, és ezeket napi számát ki lehessen egyensúlyozni. Egyes programokban van ilyen beállítás.

4.3. Nulladik órák

A nulladik óra általában vészmegoldás, ezért nem tartottuk elsődlegesnek a támogatását. Későbbiekben lehetséges, hogy adunk erre valamilyen megoldást. Természetesen kézi felvitellel most is hozzá lehet adni nulladik órákat.

4.4. Épületek, mozgás, átállások

Ha az iskola több épületből áll, jogos igény lehet, hogy minél kevesebbszer kelljen az épületek között átjárni. Ugyanez a gond persze jelentkezhet egyetlen épületen belül is.

Hasonló probléma, amikor bizonyos órák egymásutánisága nem szerencsés. Például két olyan óránál, ahol átöltözés (testnevelés), eszközfelállítás (fizika, kémia) vagy egyéb előkészület illetve utómunka szükséges.

Ezek alapvetően optimalizációs kérdések, kezelésüket a gyenge feltételekre adott komplex megoldással együtt tervezzük.