بسمه تعالی

صابر بیجستانی مقدم

استاد : دکتر ایوب باقری

موضوع : زمانبندی دروس دانشگاهی به روش جست و جوی تابو وفقی

عنوان مقاله : Adaptive Tabu Search for Course Timetabling

مقدمه

مقاله مورد بررسی برای حل مساله زمانبندی دروس دانشگاهی الگرورتم جست و جوی تابو وفقی را ارائه کرده است که این الگروزیتم از 3 فاز تشکیل میشد. یافتن جواب اولیه (initialization) جستوجوی دقیق همسایه های جواب بدست آمده (intensification) و بررسی نقاط متفاوتی از حوزه جست و جو (diversification) برای جلو گیری از به دام افتادن در جواب کمینه محلی. اولین فاز راه حل سعی در پیدا کردن پاسخی معتبر - که تمام محدودیت های غیرقابل اغماض (hard constraint) مساله را رعایت کرده باشد - برای مساله با استفاده از الگوریتم های حریصانه می کند و بعد از رسیدن به جوابی معتبر اقدام به کاهش هزینه های مربوط به محدودیت های قابل اغماض (soft constraint) میکند و در ادامه برای آن که یافتن واسخ فقط محدود به جستوجو در محدوده ای کوچک از فضای مساله نشود در فاز سوم (diversification) اقدام به جستوجو در نقاط دورتری از فضای بررسی شده فعلی می­شود.

معرفی مساله

مساله زمانبندی دورس دانشگاهی متشکل است از زمانبندی مجموعه از دروس در یک جدول زمانی هفتگی به طوری که تمام جلسات دروس مورد نظر به زمان و کلاس خاصی تحت محدودیت های موجود در مساله انتساب پیدا کنند. مساله زمانبندی دروس شامل 4 محدودیت غیر قابل اغماض (H1~H4) میباشد که یک زمانبندی معتبر باید تمام این 4 محدودیت را به طور کامل رعایت کرده باشد. در مجموع زمانبندی معتبر علاوه بر رعایت این 4 محدودیت اساسی 4محدودیت قابل اغماض (S1~S4)را نیز تا جای امکان با کمترین هزینه ممکن رعایت میکند.

4 محدودیت اساسی عبارتند از :

H1 : تمام جلسات دروس باید به زمان و کلاس خاصی منتسب شوند.

H2 : نباید هیچ دو درسی در یک زمان و مکان قرار بگیرند.

H3 : دروسی که در یک برنامه درسی قرار دارند و یا استاد مشابهی دارند نباید در یک زمان برگزار شوند.

H4 : اگر استاد درسی در زمان خاصی در دسترس نباشد نباید در آن زمان درس برگزار شود.

4 محدودیت غیر اساسی عبارتند از:

S1 : کلاس انتخاب شده باید گنجایش دانشجویان آن درس را داشته باشد.

S2 : تمام جلسات درس در صورت امکان در یک کلاس ثابت برگزار شوند و در غیر این صورت تا جای ممکن از تعداد کلاس های متفاوت کم باشد.

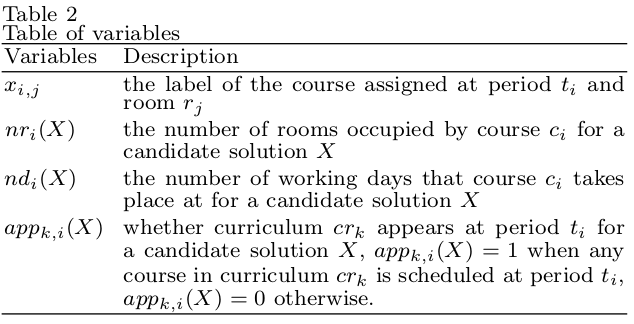
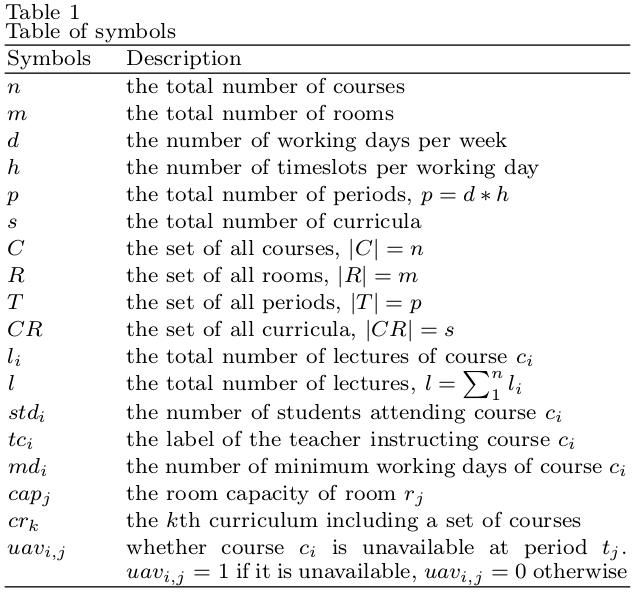
S3 : جلسات درس باید در حداقل بازه زمانی مشخص شده پخش شوند و بین جلسات فاصله زمانی مناسب رعایت شود.

S4 : زمانبندی دروس مشترک در برنامه درسی دانشجویان به گونه ای باشد که دروس با فاصله از هم برگزار نشوند و به بیان دیگر درس ایزوله ای در برنامه قرار نگیرد. منظور از درس ایزوله درسی است که ساعت قبل یا بعد آن دانشجو درسی نداشته باشد.

فرمول مساله

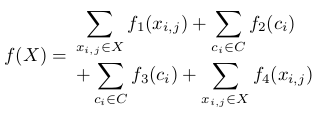
برای زمانبندی دروس از جدولی استفاده میکنیم که سطر های افقی این متناسب با کلاس های موجود و ستون های عمودی آن متناسب با زمان تشکیل درس میباشد و بدین صورت محدودیت اساسی 2 که به معنی عدم تشکیل دو کلاس در یک زمان و مکان مشترک میباشد همواره رعایت خواد شد. برای رعایت دیگر محدودیت ها نیز در ادامه فرمول های متناسب با هرکدام ارائه شده است.

علائم و متغیر های بکار رفته در فرمول های مورد استفاده برای حل مساله در دو جدول زیر آورده شده اند.



|  |  |
| --- | --- |
| توضیح | فرمول |
| در صورتی که تمام جلسات زمانبندی شده باشند H1 مقدار صفر خواهد داشت. |  |
| مدلسازی مساله به گونه ای است که این محدودیت همواره رعایت می­شود. | H2 |
| مشخص میکند که هیچ دو درسی نباید در یک برنامه درسی باشند ویا استاد مشترک داشته باشند. |  |
| اگر درس k در زمان i قرار گرفته باشد می­بایست معیار در دسترس نبودن درس k در زمان i صفر باشد به این معنا که درس در این زمان در دسترس است. |  |
| در صورتی که گنجایش کلاس کمتر از تعداد دانشجویان درس باشد به نسبت تعداد دانشجویان مازاد بر ظرفیت کلاس میزان هزینه این محدودیت افزایش خواهد داشت.  1=1α |  |
| اگر جلسات یک درس در بیش از یک کلاس برگزار شوند هزینه متناسب با نقض این محدودیت نیز افزایش خواهد داشت.  2=1α |  |
| در صورتی که تمام جلسات درس در تعداد روز کمتری نسبت به حداقل روز کاری مشخص شده برای درس برگزار شوند این محدودیت هزینه ای متناسب با شدت نقض این محدودیت خواهد داشت.  3=5α |  |
| در صورت مشاهده درس ایزوله در هریک از گروه های برنامه های درسی میزان هزینه ناشی از نقض این محدودیت افزایش میابد.  4=2α |  |

روش پینهادی ارائه شده در مقاله مورد مطالعه برای حل مساله به گونه ای است که در ابتدا اقدام به یافتن راه حلی با رعایت کامل محدودیت های اساسی میکند و لذا هزینه مربوط به این محدودیت ها همواره صفر خواهد بود و در ادامه سعی به کاش میزان هزینه ناشی از نقض محدودیت های غیر اساسی میکند. که تابع هزینه مورد بررسی برای مقایسه وضعیت راه حل بدست آمده با بهترین راه حلی که تا کنون برنامه بدست آورده است به شکل زیر می­باشد. که توابع f1 تا f4 همان هزینه های متناسب با محدودیت های غیر اساسی یک تا چهار می باشند.



(1)

یافتن جواب اولیه (initialization) :

هدف این مرحله همانطور که قبلا گفته شد بدست آوردن یه راه حل صرفا سازگار با محدودیت های اساسی مساله می باشد که برای رسیدن به این پاسخ از روشی مبتنی بر هیورستیک حریصانه استفاده میشود که به طور متوالی تا زمانی که تمام جلسات دروس زمانبندی بشوند ادامه میابد. در هر بار اجرا دو عملیات مجزا انجام خواهد شد. اولی انتخاب کی از دروسی که هنوز بطور کامل زمانبندی نشده و دیگری انتساب یک از جلسات درس انتخاب شده به یک زمان و کلاس مناسب و در نهایت قرار دادن آن در جدول برنامه درسی. به این منظور از دو هیورستیک استفاده میشود بطوری که جلسه درس به روشی پویا انتخاب و زمانبندی میشود با این تفکر که ابندا به سراغ زمان هایی میرویم که کمترین دسترسی به آنها ممکن باشد. به این معنی که دروس کمتری قابلیت برگزاری در آن بازه های زمانی را دارند. توجه داشته باشید که هزینه های مربوط به نقض محدودیت های غیر اساسی نیز در انتخاب های ما مورد بررسی قرار می گیرند. برای بیان بهتر این مرحله لازم است برخی علائم مورد استفاده در الگوریتم معرفی شود.

X : جدول زمانی ناقص بدست امده تا این لحظه

apdi(X) : تعداد کل بازه های زمانی قابل دسترس برای درس Ci  تحت برنامه X.

apsi(X) : تعداد کل موقیعت های (زمان-کلاس) در دسترس برای درس Ci تحت برنامه X.

nli(X) : تعداد جلسات زمانبندی نشده درس Ci تحت برنامه X.

uaci,j(X) : تعداد کل جلسات باقی مانده زمانبندی نشده ای که در صورت انتساب یک جلسه از درس Ci در زمان tj دیگر نمی توانند در زمان tj قرار بگیرند.

برای انتخاب یک درس از میان درس های زمانبندی نشده از هیورستیکی که در ادامه آمده است (HR1)استفاده می­شود.

1. انتخاب درس با کمترین مقدار apdi(X)/
2. در صورت انتخاب چند درس با کمترین مقدار یکسان ، اکنون درس با کمترین مقدار apsi(X)/ انتخاب می­شود.
3. اگر همچنان بیش از یک درس با شرایط یکسان وجود داشت درس با بیشترین مقدار confi انتخاب می­شود که confi بیانگر تعداد دروسی است که با درس Ci  دانشجویان یا اساتید مشترک دارند.

کد تابع HR1()

/// <summary>

/// درسی که مطابق شرایط هیورستیک 1 در مقاله باشد را پیدا میکند

/// </summary>

/// <returns>اندیس درسی که باید زمانبندی شود را برمی گرداند</returns>

public int HR1()

{

double hr1 = 0;

double hrmin = double.MaxValue;

List<int> cids = new List<int>();

for (int i = 0; i < apd.Length; i++)

{

if (nl[i] > 0)

{

hr1 = apd[i] / Math.Sqrt(nl[i]);

if (hr1 < hrmin)

{

hrmin = hr1;

cids.Clear();

cids.Add(i);

}

if (hr1 == hrmin)

{

cids.Add(i);

}

}

}

if (cids.Count == 1) return cids.First();//HR1(i)

hrmin = double.MaxValue;

List<int> cids2 = new List<int>();

foreach (int item in cids)

{

hr1 = aps[item] / Math.Sqrt(nl[item]);

if (hr1 < hrmin)

{

hrmin = hr1;

cids2.Clear();

cids2.Add(item);

}

if (hr1 == hrmin)

{

cids2.Add(item);

}

}

if (cids2.Count == 1) return cids2.First();//HR1(ii)

cids.Clear();

int hrmax = -1;

int cindex = cids2.First();

foreach (var item in cids2)

{

string cid = LC[item].CourseID;

hr1 = 0;

foreach (var conf in p.Curricula)

{

foreach (var memConf in conf.Courses)

{

if (memConf == cid)

{

hr1++;

}

}

}

if (hr1 > hrmax)

{

hrmax = (int)hr1;

cindex = item;

}

}

return cindex; //HR1(iii)

}

در این هیورستیک دروس با کمترین بازه زمانی دردسترس و بیشترین جلسه زمانبندی نشده اولویت بیشتری دارند. حال که درس مورد نظر برای زمانبندی انتخاب شد (فرض کنید درس Ci\*) می­بایست زمانی را برای این درس انتخاب کنیم که در گام های بعدی دروس کمتری به آن زمان علاقه داشته باشند که به این منظور از هیورستیکی که در ادامه آمده است (HR2) برای انتخاب زمان- مکان مناسب استفاده می­شود.

برای هز جفت زمان – مکان در دسترس (tj و rk) تلاش میکنیم تا کمترین مقدار تابع وزنی زیر را بدست آوریم:

G(j , k)= k1 \* uaci\*,j(X) + k2 \* Δfs(i\*,j,k) ; k1= 1.0 , k2=0.5 (2)

که تابع Δfs(i\*,j,k) میزان هزینه ناشی از نقض محدودیت های غیر اساسی ، بواسطه­ی انجام عملیات انتساب <C­i\*,tj,rk> می­باشد.

کد تابع HR2()

/// <summary>

/// زمان - مان متانسب با هیورستیک 2 مقاله را پیدا میکند

/// </summary>

/// <param name="cindex">اندیس درس انتخاب شده برای انتساب زمان - مکان</param>

/// <returns>[0]: period [1]: room </returns>

public int[] HR2(int cindex)

{

k1 = 1;

k2 = 0.5;

int[] pos = new int[2];

pos[0] = -1;

pos[1] = -1;

List<int> uavtimes=new List<int>();

uavtimes = getUavPeriods(cindex);

double minCost = double.MaxValue;

int[] bestPos = new int[2];

bestPos[0] = -1;

bestPos[1] = -1;

for (int t = 0; t < p.Periods\_per\_day\*p.Days; t++)

{

if (!uavtimes.Contains(t))

{

for (int r = 0; r < p.Rooms.Length; r++)

{

if(timeTable[t][r].CourseID==null){

int[] pr = new int[2];

pr[0] = t / p.Days;

pr[1] = t % p.Periods\_per\_day;

solution tempSol = new solution();

solutionCopier.copy(this, out tempSol);

tempSol.insertLecture(cindex, t, r);

double cost = k1 \* uac(cindex, pr) + k2 \* validator.getCost(p, tempSol);

if (cost < minCost)

{

minCost = cost;

bestPos[0] = t;

bestPos[1] = r;

}}

}

}

}

return bestPos;

}

با توجه به توضیحات داده شده شبه کد اولین فاز از الگوریتم ارائه شده در مقاله به صورت زیر خواهد بود.

|  |  |
| --- | --- |
| شبه کد ارائه شده در مقاله | کد استفاده شده در برنامه |
|  | solution X0 = new solution();  int[] pos = new int[2];  do  {  int ci=X0.HR1();  pos = X0.HR2(ci);  X0.insertLecture(ci, pos[0], pos[1]);  } while (X0.LC.Count > 0 && pos[0] != -1);//end of part1: initialization |

جست و جوی تابو

بعد از بدست اوردن یک جواب سازگار برای مساله نوبت آن میرسد که این جواب تا جای امکان بهینه شود که به این منظور از جست و جوی تابو استفاده شده است. در الگوریتم جست و جوی تابو ماهیت همسایه ها اهمیت خیلی بالایی دارد که در مساله ای با آن مواجه هستیم همسایه ها شامل حرکت های قابل انجام روی برنامه درسی بدست امده می­باشند به این صورت که هر حرکت به معنای انتقال یک درس از یک زمان – مکان به زمان – مکان جدیدی میباشد. در الگوریتم ارائه شده در مقاله 2 نوع همسایگی تعریف شده است که عملیات جست و جوی تابو به صورت متوالی روی این دو نوع همسایگی اعمال میشود به این صورت که یک بار روی همسایه های از نوع اول جست و جو انجام می­شود و سپس نتیجه این مرحله با جست و جو در همسایگی های نوع 2 ادامه میاد و مجدد نتیجه روی همسایگی های نوع 1 و این روند ادامه دارد تا جایی که پیشرفتی حاصل نشود.

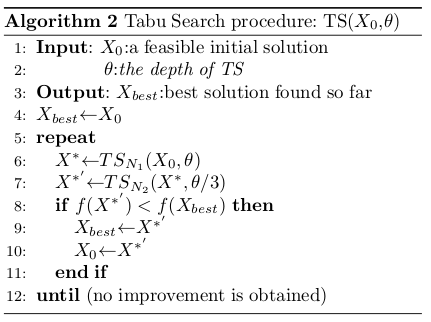
همسایگی های نوع 1 :

این نوع از همسایگی ها شامل جابجا کردن 2 درس در دو موقیعت متفاوت از برنامه زمانی با یکدیگر میباشد. که البته جابجا کردن یک درس با یک خانه خالی از برنامه نوع خاصی از این انتقال به حساب می­آید.

همسایگی های نوع 2 :

در این نوع همسایگی فقط زمان دروس جابجا میشود و تغییر در مکان آنها انجام نمیشود .

شبه کد ارائه شده در مقاله :



کد تابع جست و جوی تابو

/// <summary>

/// tabu search

/// </summary>

/// <param name="X">a feasible initial solution</param>

/// <param name="theta">θ:the depth of TS</param>

/// <returns>X best :best solution found so far</returns>

public solution TS(solution X,int theta)

{

int loops = theta;

solution Xbest = new solution();

solutionCopier.copy(X, out Xbest);//make a deep copy of X to Xbest

simpleSwapList=new List<int[]>();

poindex = 0;

int r1, r2, t1, t2;

int len = X.timeTable.Length \* X.timeTable[0].Length;

for (int i = 0; i < len; i++)

{

for (int j = i + 1; j < len; j++)

{

// simswap[0]:t1

// simswap[1]:r1

// simswap[2]:t2

// simswap[3]:r2

int[] simswap = new int[4];

t1 = simswap[0] = i / X.timeTable[0].Length;

r1 = simswap[1] = i % X.timeTable[0].Length;

t2 = simswap[2] = j / X.timeTable[0].Length;

r2 = simswap[3] = + j % X.timeTable[0].Length;

string course1 = X.timeTable[t1][r1].CourseID;

string course2 = X.timeTable[t2][r2].CourseID;

simpleMove m1 = new simpleMove();

simpleMove m2 = new simpleMove();

m1.courseId = course1;

m1.r = r2;

m1.t = t2;

m2.courseId = course2;

m2.r = r1;

m2.t = t1;

if( isAvl(X,m1) && isAvl(X,m2) )

{

if (m1.courseId != null || m2.courseId != null)

simpleSwapList.Add(simswap);

}

}

}

do

{

solutionCopier.copy(TSN1(X, theta),out Xstar);

solutionCopier.copy(TSN2(Xstar, (int)(theta / 3)),out XstarPrime);

double costXstarPrime, costXbest;

costXbest = validator.getCost(prblm, Xbest);

costXstarPrime = validator.getCost(prblm, XstarPrime);

if (costXstarPrime < costXbest)

{

solutionCopier.copy( XstarPrime,out Xbest);

solutionCopier.copy( XstarPrime,out X);

}

} while (loops-->0);

return Xbest;

}//end of TS

به منظور جلو گیری از به دام افتادن در کمینه محلی از فاز سوم الگوریتم که وظیفه آن حرکت به فضای دیگری از فضای جست و جو می­باشد استفاده می­کنیم. که این عمل شانس مارا در رسیدن به نقاط کمینه احتمالی دیگر افزایش میدهد. البته اگر این انتقال خیلی زیاد باشد ممکن است جواب های بهینه ای را از دست بدهیم و در انتخاب شدت جابجایی در محیط جست و جو از روشی پویا استفاده میکنیم که متغیر η بیان کننده شدت این جابجایی میباشد. که مقدار آن از رابطه زیر محاسبه میشود:

η←max{ηmin + λ · ξ, ηmax {

λ = 0.3

ηmin = 4

ηmax = 15

ξ = نعداد دفعاتی که به طور متوالی موفق به کاهش میزان تابع هزینه نسبت به بهترین جواب کنونی بدست آمده نشده ایم