Virtual Academic Advisor – Engine Development Document

Written in C#

Andrue Cashman

University of Washington - Bothell

Advisor: Dr. Erika Parsons

December 16th, 2018

[About this Document 5](#_Toc532823249)

[Version History 5](#_Toc532823250)

[Big Picture 6](#_Toc532823251)

[Relationship with UI, API, and DB 6](#_Toc532823252)

[VSA Database Calls 6](#_Toc532823253)

[A Note on JSON 7](#_Toc532823254)

[CourseNetwork Class 7](#_Toc532823255)

[Overview 7](#_Toc532823256)

[Variables 7](#_Toc532823257)

[Constructors 7](#_Toc532823258)

[Public – BuildNewtork() 7](#_Toc532823259)

[Private – BuildIndex() 8](#_Toc532823260)

[Public – FindShortPath(Int) 8](#_Toc532823261)

[Public – Display Functions 9](#_Toc532823262)

[CourseNode Class 9](#_Toc532823263)

[Overview 9](#_Toc532823264)

[Variables 9](#_Toc532823265)

[Constructors 10](#_Toc532823266)

[MakeNewList() 10](#_Toc532823267)

[DayTime Class 10](#_Toc532823268)

[Overview 10](#_Toc532823269)

[Variables 10](#_Toc532823270)

[Constructors 10](#_Toc532823271)

[Getters 11](#_Toc532823272)

[Setters 11](#_Toc532823273)

[Overloaded: == and != 11](#_Toc532823274)

[DBConnection Class 11](#_Toc532823275)

[Overview 11](#_Toc532823276)

[Public – DataTable ExecuteToDT(string) 11](#_Toc532823277)

[Public – String ExecuteToString(string) 11](#_Toc532823278)

[Private – OpenSQLConnection() 11](#_Toc532823279)

[DegreePlan Class 11](#_Toc532823280)

[Overview 11](#_Toc532823281)

[Constructor 12](#_Toc532823282)

[Getter 12](#_Toc532823283)

[Driver Class 12](#_Toc532823284)

[Overview 12](#_Toc532823285)

[Method 1: Manual Run 12](#_Toc532823286)

[Method 2: Automated Run 12](#_Toc532823287)

[Job Class 12](#_Toc532823288)

[Overview 12](#_Toc532823289)

[Variables 12](#_Toc532823290)

[Constructors 13](#_Toc532823291)

[Getters 13](#_Toc532823292)

[Setters 13](#_Toc532823293)

[Overloaded: ==, !=, and Equals(object) 13](#_Toc532823294)

[Machine Class 13](#_Toc532823295)

[Overview 13](#_Toc532823296)

[Variables 13](#_Toc532823297)

[Constructors 14](#_Toc532823298)

[Public – canDoJob(Job) 14](#_Toc532823299)

[Getters 14](#_Toc532823300)

[Setters 14](#_Toc532823301)

[Primary Functions 14](#_Toc532823302)

[Overloaded: ==, !=, Equals(object), Equals(Machine) 14](#_Toc532823303)

[Display Methods 15](#_Toc532823304)

[MachineNode Class 15](#_Toc532823305)

[Variables 15](#_Toc532823306)

[Constructors 15](#_Toc532823307)

[Getters 15](#_Toc532823308)

[Mutators 15](#_Toc532823309)

[ParameterSet Class 16](#_Toc532823310)

[Overview 16](#_Toc532823311)

[Variables 16](#_Toc532823312)

[Getters 16](#_Toc532823313)

[Preferences Class 17](#_Toc532823314)

[Overview 17](#_Toc532823315)

[Variables 17](#_Toc532823316)

[Inner Class: CourseNumbers – No longer functional as of version 2.2 17](#_Toc532823317)

[Constructors 17](#_Toc532823318)

[Deserialize() 18](#_Toc532823319)

[Getters 18](#_Toc532823320)

[Data Interpretation 18](#_Toc532823321)

[Scheduler Class 18](#_Toc532823322)

[Overview 18](#_Toc532823323)

[Variables 18](#_Toc532823324)

[Constructors 19](#_Toc532823325)

[Set Up 19](#_Toc532823326)

[Scheduling Algorithm 21](#_Toc532823327)

[Results 22](#_Toc532823328)

# About this Document

This document is not intended to go into exacting detail of every aspect of the Scheduling algorithm or its various parts. The purpose of this document is to serve as a reference to and give short descriptions of the functions themselves. This document also helps in declaring ownership of each portion of written code to some degree should questions arise. This document is intended to be a living document for which each individual who works on the algorithm will also add to this document to reflect those changes. This document further establishes a central repository of diagrams and information regarding the overall relationship of all parts of the algorithm. If you are looking for specific documents regarding an individual’s sole efforts please refer to their own documents on the matter:

* Eremenko, P. “Virtual Academic Adviser Scheduling Algorithm Design Document”. Dr. Parsons. Spring 2018
* Cashman, A. “Virtual Student Advisor Algorithm Analysis and Refinement”. Dr. Parsons. Winter 2018.

# Version History

* 2.2: With the latest update to the database the columns that once contained CompletedCourses and PlacementCourses have been removed from the ParameterSet table. The algorithm has been adjusted to accommodate this. However, MakeStartingPoint() does not do anything. Manual entry of the starting courses needs to be used until CompletedCourses and PlacementCourses are reimplemented.
* 2.15: Most data has been switched to variables instead of being hardcoded.
* 2.1: Algorithm is now running with live connections to the VSA Database
* 2.0: Algorithm completed with preferences and usable data hardcoded to algorithm

# Big Picture

For the time being there are several distinct parts of the algorithm that need to be mentioned in advance to understand better the algorithm.

## Relationship with UI, API, and DB

This algorithm is intended to be invoked by an API that interprets the output of this algorithm and packages it to the UI. While initial parameters are required by the API for this algorithm to function, the algorithm is designed to frequently make contact with the Virtual Student Advisor Database and any changes there must be cross-referenced with the algorithm to ensure desired behavior. It should be said that it is not expected or anticipated behavior for the UI to invoke commands directly from this algorithm.

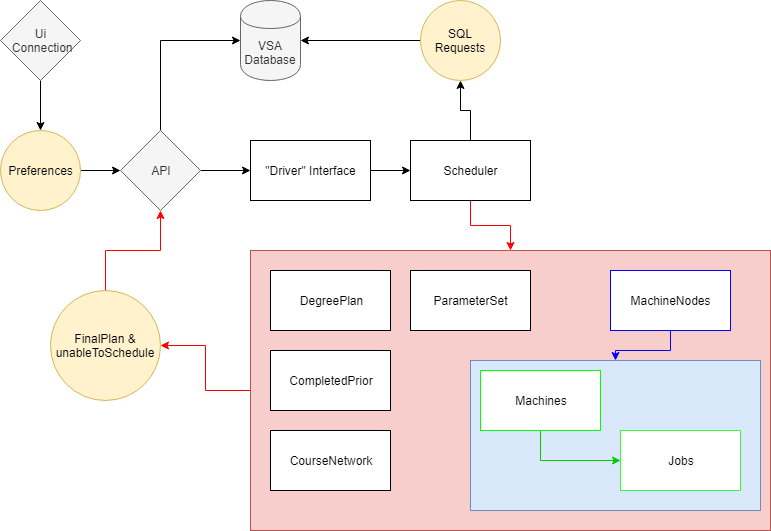


Figure : Relationship Model of UI, DB, API, & Algorithm

## VSA Database Calls

|  |  |
| --- | --- |
| **Class Containing Call** | **Table Referenced** |
| Preferences.cs | ParameterSet |
| Preferences.cs | Course |
| Scheduler.cs | Prerequisite |
| Scheduler.cs | Course |
| Scheduler.cs | AdmissionRequiredCourses |
| Scheduler.cs | CourseTime |

## A Note on JSON

This algorithm uses Newtonsoft.Json to serialize and deserialize information from the database frequently. Additionally, not included in this document but used throughout the source code or identifiers that Newtonsoft uses to help in the serialization and deserialization processes. These identifiers are [JsonProperty] and [JsonIgnore]. Descriptions of each of these are easily accessible on Google but suffice to say that from this point forward they will not be mentioned again.

# CourseNetwork Class

## Overview

The Course Network class is designed to represent all courses in a given institution and provide a means of determining each class’s immediate prerequisites. This process is assisted by a list of integers, which serves as a hash table, to make the process much faster than performing multiple linear searches through the CourseNode list.

Primary Author: Andrue Cashman

## Variables

String allCourses

Used to store the serialized JSON object of all the courses in the database

String prereqs

Used to store the serialized JSON object that connects a course to its prerequisites

### List<CourseNode> courseNetwork

The structure resulting from the combination of the deserialized JSON objects of both allCourses and prereqs. Essentially, each element in this list contains a single course and also a list of its direct prerequisites.

List<int> courseIndex

Provides a means to perform an O(1) lookup of each course in the courseNetwork List.

## Constructors

CourseNetwork(string, string)

Each string is the serialized JSON objects representing all courses an institution offers and each courses’ prerequisites respectively.

## Public – BuildNewtork()

Deserializes allCourses into the courseNetwork to provide an initial list of all courses. Afterwards, prereqs is deserialized into its own, temporary, CourseNode list. A call to BuildIndex() builds the hash table that provides quick access to each course in CourseNetwork.

After the initial setup, each element in the temporary list representing the prerequisites is added to the prerequisite list for its respective course in courseNetwork.

## Private – BuildIndex()

Builds a List of Integers where the index of each element is a course ID that exists in the Course Network. The value of each element in this list of integers is the index of the corresponding course in the courseNetwork list.

Because of the assumption that neither JSON object is in any particular meaningful order the index will grow to the size of the largest course ID. For example; if there is a course ID that is 7986, and it is the second element in courseNetwork, there will be 7986 total elements in courseIndex. This also leaves open the possibility that there will be many elements in courseIndex that will have values of null.

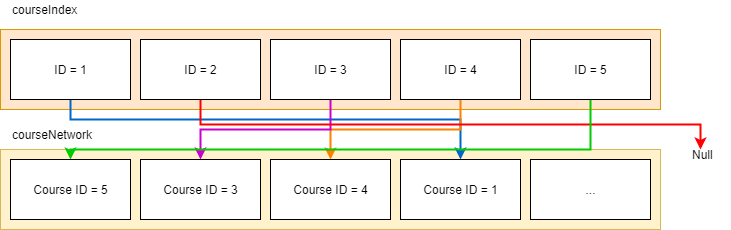


Figure : courseIndex and courseNetwork Relationship

## Public – FindShortPath(Int)

This function does the work of collecting the direct prerequisites of a specific course and then returns the resulting list back to the calling function (e.g. Main()). This function groups each prerequisite to its respective group and assigns them to an index. This is to represent the differing “paths” that can be taken to reach a particular course. This solves the AND and OR situations for a course’s requirements.

For example, in order to take the course ‘Z’ a student can take courses A AND B OR courses A AND C OR course D. Each individual path is valid, but an entire path needs to be “completed” to satisfy the requirements.

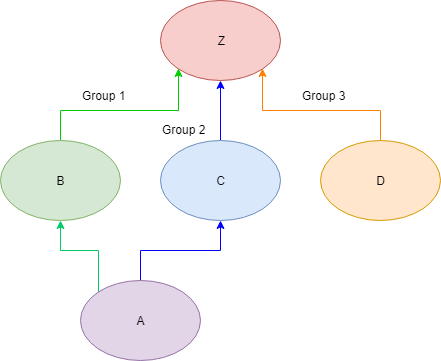


Figure : Prerequisite Group Paths

## Public – Display Functions

### ShowNetwork()

Displays the contents of the courseNetwork List by showing the element’s course ID and then the course IDs of the prerequisite list

### ShowShortList(List<CourseNode>)

Displays the list contents of the given List. Intended for use in displaying the results of FindShortPath(int courseID).

# CourseNode Class

## Overview

Provides a structure to be used by CourseNetwork in the implementation of showing the connections between a course and its direct prerequisites.

Primary Author: Andrue Cashman

## Variables

### Int courseID

Represents a course’s unique ID, as given by the database.

### Int groupID

Represents the grouping of this course to a prerequisite group path. With the exception of describing prerequisites, this variable is not used.

### Int prerequisiteID

Represents the course ID of a prerequisite. With the exception of building the courseNetwork, this variable is not used.

### List<CourseNode> prereqs

Describes this course’s prerequisites.

## Constructors

### Default

Sets all values to zero and the prereq list to Null

### CourseNode(CourseNode, Bool)

Sets all values to the values of the given CourseNode. The given Boolean determines if the prereq list is to be copied or not.

## MakeNewList()

Sets up the prereq list as a new List<CourseNode> if it wasn’t set up already.

# DayTime Class

## Overview

This class is used to describe the day, starting time, and ending time of a particular course. As given the nature of courses, their can be more than one DayTime object per Course.

Primary Author: Polina Eremenko

## Variables

### Int day

The integer representation of a day as given by the WeekDay table in the VSA Database. With 1 representing Monday and 7 representing Sunday.

### Int start\_time

The integer representation of a course’s starting time as given by the TimeSlot table in the VSA Database. A value of 1 represents the time 00:00:00 and a value of 144 represents the time 23:50:00.

### Int end\_time

The integer representation of a course’s starting time as given by the TimeSlot table in the VSA Database. A value of 1 represents the time 00:00:00 and a value of 144 represents the time 23:50:00.

## Constructors

### Default

Sets all values to zero.

### DayTime(int, int, int)

Sets the values of day, start\_time, and end\_time to those provided by the given values respectively.

## Getters

GetDay(), GetStartTime(), and GetEndTime() return their respective values.

## Setters

SetDay(int), SetStartTime(int), and SetEndTime(int) set the value to the given respective values.

SetDayTime(int, int, int) sets all values of this DayTime to the given values.

## Overloaded: == and !=

The comparison operators are overloaded to compare the equality of two DayTime objects. The left-hand side’s day, start\_time, and end\_time are compared to the respective values of the right-hand side.

# DBConnection Class

## Overview

Provides a centralized means to connect to the Virtual Student Advisor Database. The only variable of this class is an SqlConnection.

Primary Author: Andrue Cashman

## Public – DataTable ExecuteToDT(string)

This function opens a connection to the VSA database and then executes the given string as a SQL query. It is assumed that the string is a valid SQL query. The results of the query are passed back to the calling function (e.g. Main) as a DataTable object.

## Public – String ExecuteToString(string)

This function opens a connection to the VSA database and then executes the given string as a SQL query. It is assumed that the string is a valid SQL query and is expecting the query to resolve as a JSON object. The results of the query are passed back to the calling function (e.g. Main) as a serialized JSON string.

## Private – OpenSQLConnection()

This function initializes a SqlConnection with the credentials required to connect to the VSA Database. If, for any reason, the credentials to the VSA database were to change, this is where those credentials would need to be updated.

# DegreePlan Class

## Overview

This class is used to describe the required courses of a particular degree with consideration to the intended school the student wishes to transfer to. The only variable is a List<List<Job>> to represent the list of courses that are required.

Although not used, this class has been made extensible to accommodate multiple Lists of required courses.

Primary Author: Polina Eremenko

## Constructor

### DegreePlan(List<Job>)

Assigns the given List<Job> to this DegreePlan.

## Getter

GetList(int) returns the list of required courses at the given index.

# Driver Class

## Overview

This class is intended to operate the Scheduling Engine; instantiating and returning output. The purpose is to deliver the output as a serialized JSON string to be interpreted by the API.

Primary Author: Andrue Cashman, Polina Eremenko

## Method 1: Manual Run

This method builds the scheduler with manual entries of the desired plan length and summer preference with additional function calls required to set up the starting point and the required degree plan

## Method 2: Automated Run

This method builds the scheduler with the passed integer referring to a ParameterSet entry in the VSA Database to build all the required information to make the Scheduler run. This is currently, the easiest and cleanest way to run the scheduler.

# Job Class

## Overview

This class is used to represent a course and its status in the scheduling algorithm.

Primary Author: Polina Eremenko

## Variables

### Int id

The ID of the course.

### Bool scheduled

Represents whether or not this Job has been scheduled.

### Bool prerequisitesScheduled

Represents whether or not this Job’s prerequisites have been scheduled

### Int quarterScheduled

The quarter in which the Job was scheduled, if applicable.

### Int yearScheduled

The year in which the job was scheduled, if applicable.

## Constructors

### Default

Initializes all integer variables as zero and all booleans to false.

### Job(int)

Sets the ID of this job to the given integer value. Sets the booleans to false and the other integers to -1.

## Getters

GetScheduled(), GetPrerequisitesScheduled(), GetQuarterScheduled(), GetYearScheduled(), and GetID() return the values of the respective variables of this job.

## Setters

SetPrerequisitesScheduled(bool), SetScheduled(bool), SetQuarterScheduled(int), and SetYearScheduled(int) set the respective variables to the given values.

There is no setter for a Job’s ID.

## Overloaded: ==, !=, and Equals(object)

The equality operators are overloaded to first check if the object is being compared to itself and then compares the IDs of the left-hand Job with that of the right-hand job.

# Machine Class

## Overview

This class represents a course as a machine. Having a course described in this way, in addition to the Job class, allows the algorithm to perform specific functions, such as manipulating a schedule, while keeping the integrity of the Job class intact. Additionally, the Machine class adds an additional layer of information with the inclusion of the DateTime class.

Primary Author: Polina Eremenko

Updated by: Andrue Cashman (Equality Operators)

## Variables

### Bool inUse

Represents the status of the Machine in the Scheduling algorithm. This presents unnecessary repetition in the algorithm, such as attempts at scheduling the same machine twice.

### Int year

The year that the Machine represents

### Int quarter

The quarter that the Machine represents

### List<Job> jobs

Represents the list of courses residing on this machine.

### Job currentJobProcessing

Used as the main ID of the Machine, specifically the ID of a Course and whether or not it or its prerequisites have been scheduled.

### List<DateTime>

The list of available DateTimes that the job is being offered for a particular quarter and year

## Constructors

### Default

Sets year and quarter to zero, inUse to false, currentJobProcessing to null, and initializes dateTime and jobs as new (empty) Lists.

### Machine(int, int, List<DateTime>, List<Job>)

Sets year and quarter to the values of the first two integers respectively. Assigns dateTime to the given DateTime List and jobs to the given Job List. InUse is set to false and currentJobProcessing remains null.

### Copy Constructor: Machine(Machine)

Performs a variable by variable deep copy of the given Machine to the present Machine.

## Public – canDoJob(Job)

Checks if the given Job is on this Machine’s List<Job>. Used by the algorithm to determine if the given job should be scheduled.

## Getters

CheckInUse(), GetYear(), GetQuarter(), GetCurrentJobProcessing(), GetDateTime() returns the values of their respective variables.

There is no getter for the variable jobs.

## Setters

SetInUse(bool), SetYear(int), SetQuarter(int), SetCurrentJobProcessing(Job) sets the values of their respective variables to the given values.

There are no direct setters for jobs or dateTime.

## Primary Functions

### Public – Add/Remove Job

Adds a job to this Machine’s List<Job>. DeleteJob(Job) removes a job from this Machine’s List<Job>.

### Public – AddDayTime(DayTime)

Adds a DayTime to this Machine’s List<DayTime> if it is not present in the machine already.

## Overloaded: ==, !=, Equals(object), Equals(Machine)

Defines the equality operators to check for self-referencing before comparing the years, quarters, and List<DayTime> of the two Machines being compared.

### Private – ContainsDayTime(List<DayTime>, DayTime)

Returns a Boolean based on whether or not a given DayTime is present amongst a List of DayTimes. Used as a helper function for the equality operators.

## Display Methods

PrintBusyMachine() provides a formatted means of viewing the values of the Machine.

# MachineNode Class

The primary purpose of this class is to house a List<Machine> to represent individual quarters of a study plan.

Primary Author: Polina Eremenko

## Variables

### Int creditsScheduled

Keeps track of the total amount of credits scheduled on the current node – Not Implemented

### Int majorCreditsScheduled

Keeps track of the amount of credits required by the major – Not Implemented

### List<Machine> machines

Represents the list of courses present on each MachineNode

### Int year

Represents the year of the MachineNode

### Int quarter

Represents the quarter of the MachineNode

### Int classesScheduled

Represents the amount of classes that have been scheduled on this MachineNode

## Constructors

### MachineNode(int, int)

Initializes the MachineNode with the given year and quarter, respectively. Also, sets all other integers to zero and initializes an empty List for the List<Machine>.

## Getters

GetYear(), GetQuarter(), GetMachines(), GetCreditsScheduled(), GetClassesScheduled(), and GetMajorCreditsScheduled() return the values of their respective variables.

GetAllScheduledmachines() returns a list of only Machines that have been scheduled after checking the each Machine’s inUse variable.

## Mutators

### AddClassesScheduled(int)

Adds the given value to the value of classesScheduled.

### AddMachine(Machine) & AddMachine(int, int, List<DayTime>, List<Job>)

Adds the given machine to the List<Machine> of this MachineNode.

### RemoveMachine(Machine)

Removes a machine from the List<Machine> of this MachineNode

# ParameterSet Class

## Overview

Represents the variables that exist on the ParameterSet table in the VSA Database. Any changes to the ParameterSet table need to be reflected in this class. The resulting data from the database should be used to alter the scheduling algorithm to reflect the preferences.

Primary Author: Andrue Cashman

## Variables

All variables are, and should, reflect every column referenced in the ParameterSet table of the VSA database.

|  |  |  |
| --- | --- | --- |
| Type | Name | Implemented in Algorithm |
| Int | ParameterSetID | Yes |
| Int | MajorID | Yes |
| int | SchoolID | Yes |
| Int | JobTypeID | No |
| Int | TimePreferenceID | No |
| Int | QuarterPreferenceID | No |
| DateTime | DateAdded | Not Used |
| DateTime | LastDateModified | Not Used |
| Int | NumberCoreCoursesPerQuarter | No |
| Int | MaxNumbersOfQuarters | Yes |
| Int | CreditsPerQuarter | No |
| Int | Status | Not Used |
| String | SummerPreference | Yes |
| Int | EnrollmentType | No |

## Getters

All variables, except DateAdded, LastDateModified, and Status have functions designed to return their respective values.

# Preferences Class

## Overview

Establishes a connection to the VSA database to populate a ParameterSet object to be used by the algorithm to develop a schedule.

Primary Author: Andrue Cashman

## Variables

### String JsonString

Represents the string that was used to generate the ParameterSet object

### DBConnection dbHit

Used by this class to establish a connection to the VSA database

### List<ParameterSet> prefs

Represents the preferences in the ParameterSet table of the VSA Database as a deserialized JSON object.

### List<Job> priors

Represents the list of courses that do not need to be scheduled due to already having been completed or placement tests. Generated from deserializing CompletedCourses and PlacementCourses from the ParameterSet table.

### Bool summerIntent

Represents the preference of attending summer courses in the algorithm. Interpreted from SummerPreference from the ParameterSet table.

## Inner Class: CourseNumbers – No longer functional as of version 2.2

### Overview

Used to help deserialize the contents of CompletedCourses and PlacementCourses under the assumption that they are stored as a JSON string in ParameterSet. CourseNumbers in Course from the VSA Database are represented as ENGL& 101, MATH& 141, and similar conventions. The only variable is the string that represents the CourseNumber from Course. The only function is returnCourse(), which returns the contents of the string.

## Constructors

### Default

Sets JsonString to an empty string, leaving all other variables as null.

### Preferences(string)

Assigns the given string to JsonString and then calls the deserialize function to populate the other variables.

### Preferences(int)

Establishes a connection to the VSA database which then queries the ParameterSet table the JSON representation of the given integer. An invalid JSON string will result if the given integer is beyond the domain of ParameterSet. Afterwards, SetPriors() and determineSummer() are invoked to further interpret data from ParameterSet that is not immediately usable.

## Deserialize()

Deserializes JsonString and initializes the variables of the ParameterSet object.

## Getters

getMajor(), getSchool(), getQuarters(), and getSummer() return the values of their respective variables with Major, School, and Quarters being variables belonging to ParameterSet.

## Data Interpretation

### SetPriors() – No Longer Functional as of 2.2

Initializes Priors as a new List<Job> and then invokes the addToPriors function to deserialize CompletedCourses and PlacementCourses.

### addToPriors(string) – No Longer Functional as of 2.2

Deserializes the given string into a List<CourseNumbers> object and then queries the database for each CourseNumber in the Course table to retrieve the CourseID to be then added to priors List. Additionally, each resulting Job added to priors has it’s scheduled variable set to true to prevent scheduling of that job.

### determineSummer()

Checks the first character given by the string representing SummerPreference in ParameterSet. If the first character is ‘y’ or ‘Y’ then summerIntent is set to true. Otherwise, summerIntent is set to false.

# Scheduler Class

## Overview

This class performs the primary function of scheduling courses and organizing the courses into a usable format to represent a schedule plan for a student based off of specific preferences.

Primary Author: Polina Eremenko

Updated By: Andrue Cashman (Constructors, DeterminePlanLength, InitializeMachineNodes[rewritten], MakeStartingPoint, normalizeMachines[extracted], InitDegreePlan[alternate], isScheduled[Addendum]

## Variables

### DBConnection DBPlugin

Represents that a connection to VSA Database will be made.

### CourseNetwork network

Represents the courses that are offered as well as provides a means to determine their prerequisites.

### List<MachineNode> machineNodes

Each MachineNode element represents a single quarter on a student’s study plan.

### DegreePlan myPlan

Represents the list of courses that need to be scheduled to complete the plan.

### List<Machine> finalPlan

Represents the results of the scheduling algorithm that contains the list of successfully scheduled courses. Intended as output.

### Preferences preferences

Represents the preferences that is used to operate the scheduler. Preferences include the length of a study plan, the preference to attend summer quarter, whether or not the student wants to be part- or full-time, etc.

### List<Job> completedPrior

Used as the “starting point” by the scheduler. Essentially, this contains a list of courses that are not to be scheduled.

### List<Job> unableToSchedule

Represents a list of courses that could not be scheduled by the algorithm for a variety of reasons.

### Int quarters

Represents the desired length of the study plan.

### Bool attendSummer

Represents the students desire to attend summer courses. Not currently used.

### Int yearLength

Represents how many quarters per year there should be on the study plan. Determined by summer preference.

### Int years

Represents the total years of the study plan. Derived from summer preferences and quarters.

## Constructors

### Default

Invokes the SetUp function for a study plan of 8 quarters with the intention to attend summer courses.

### Scheduler(int, bool)

Invokes the SetUp function for a study plan with the given number of quarters and the given intention to attend summer courses.

### Scheduler(int)

Invokes the SetUp function to establish a study plan with the given ID of a ParameterSet in the corresponding table on the VSA Database. Afterwards, private calls to necessary functions for the algorithm are invoked. Finally, CreateSchedule() is called to develop the study plan. This is the most automated of any of the constructors.

## Set Up

### SetUp(int, bool, int)

Collects the common functionality of three of the constructors and runs a series of functions that sets up the necessary variables for the algorithm to run.

### InitNetwork()

Establishes the CourseNetwork by making queries to the database on which courses are offered and what those courses’ prerequisites are.

### DeterminePlanLength()

Determines how many years will be on the study plan and if there is intention to attend summer courses.

### MakeStartingPoint() & MakeStartingPoint(string, string)

Establishes the List of courses that are not to be scheduled by the algorithm. The method with two given strings queries the database for the CourseIDs of the two given CourseNumbers before adding them to the list.

### InitializeMachineNodes()

Translates the total length of the study plan into a List<MachineNode> where each element is made unique by a combination of quarter and year.

### normalizeMachine()

Currently, the amount of information in CourseTime only exists for the length of one year. For this algorithm to work correctly, this information needs to be made available across all years of the study plan that the algorithm is acting on. This function takes care of that problem.

However, should there be data made available that spans multiple years, all data outside the scope of the initial year will be lost if this function is used.

The function InitMachines() establishes the data provided by CourseTime. Any change to that function can make this function incompatible with the algorithm and cause undefined behavior.

### InitDegreePlan() & InitDegreePlan(int, int)

Sets up the list of courses that need to be scheduled. Both functions query the database for the required information although one gets the preferences from Parameter Set where the other one is a invoked manually.

### planBuilder(DataTable)

Helper function to InitDegreePlan which does the actual work of adding the results from the given DataTable to the List<Job> of courses that need to be scheduled.

### InitMachines()

Makes the List<Machine> of all the courses provided by institution that can be scheduled. Two classes are different from each other if they have a different CourseID or if they have different quarter, year, or section information. All being equal, then DayTime is considered to be an extension of the course.

### addMachine()

Helper function to InitMachines, this function does the work of adding a job to an existing machine.

## Scheduling Algorithm

### CreateSchedule()

Performs an element by element scheduling of the list of courses generated by InitDegreePlan() by invoking ScheduleCourse. Then the finalPlan is generated by getting all the “busy” machines, or machines with scheduled courses, and provides the results as an output in the form of a List<Machine>.

### ScheduleCourse(Job)

A recursive function which ends at the end of a prerequisite chain or course that has already been scheduled, whichever comes first. This is accomplished by referring to the prerequisites of job in question as provided by the CourseNetwork. If prerequisites are found, then each of the prerequisites of the previous prerequisites are scheduled. When the recursive call has no more prerequisites to attempt to schedule it invokes the PutCourseOnMachine(job, group) function. If for any reason the Job, or course, was not able to be scheduled, it is then added to the unableToSchedule List.

### PutCourseOnMachine(Job, List<CourseNode>)

This function does the actual duty of scheduling the Course and placing it onto its respective MachineNode. A starting point is established at the closest quarter of the last scheduled prerequisite of the Job that is to be scheduled. If no prerequisites exist, then the job is placed in the first available machine starting at the earliest. This function also serves as the implementation point of various preferences that have yet to be implemented. These preferences, so far, are Day Preference, Time Preference, Core Course Preference, Credit/Course Limits, Job Status, and Enrollment Status.

### Various Helper Functions

### isScheduled(Job)

Checks if a job has been scheduled in the case of being already on the study plan or had been scheduled in the case of having already been completed.

### PrereqsExist(List<CourseNode>)

Checks if Prerequisites exist for the given List describing a course (see CourseNetwork). This is a good check to prevent unnecessary searches of lists.

### Overlap(Job, Machine, MachineNode)

A check to make sure that a given course does not overlap the start\_time or end\_time of any other course that has already been scheduled.

### compareDays(List<DayTime>, List<DayTime>)

A helper function for Overlap. Compares the two List of DayTimes.

### GetMostRecentPrereqs(List<CourseNode>)

Garners the quarter and year of the most recent prerequisite scheduled

### GetShortestGroup(List<CourseNode>)

Refers to the CourseNetwork to get the smallest group among prerequisites of a given course. This may not be the most optimum of functions as this does not consider a course satisfying the requirement for multiple courses’ prerequisite chains.

## Results

### List<Machine> GetBusyMachines()

This function collects all the courses that have been scheduled on each MachineNode and adds them to a List<Machine> to be used as the final study plan.