Cargo Loading

Seyed Ali Khoramshahi

Prague College
School of Computing

October 2014

Contents

introduction 0

Increasing growing in producing goods in last century has been increased the demand for finding best solution to transfer and store these products more than any time, this matter seems more important when it comes to the pollution and cost caused by all different carriers used to complete the circle of production and delivery. this is what brings us to pay more and more attention to logistics and finding best solution to optimized these procedures.

Logistics defines as the procedure of collecting, storing and delivering goods between the point of production and costumer. this can be broken down into external and internal logistics, the procedure happening inside and outside the storage place, the subjective of the following project is to offer an optimized and automatized solution for external logistic.

In order to solve the problem of external logistics it's essential to solve these two steps: first bin packing, to optimize planning of inside each carrier and then optimization of path for all carrier to find the best solution.

The result of this project leads to a cross-platform application which by given information can suggest the paths for collect or delivering the goods and a suggested planning for placing the goods inside each carrier.

Methodology 1

As mentioned in introduction current applications is consist of two main problems: first bin packing, and path optimizer.

1.1 Bin packing

Through the history of computer science the problem of bin packing has been discussed in all different variety such as different dimensions for the environment like 2-dimension or 3-dimension, and there has been so many suggested solution to this problem. although there are two main algorithm:

First fit This Heuristic algorithm is based on the ordering the package arbitrary and try to fit them into first bin if possible. After putting each packet the rest of the space in the bin will be define as a new bin appropriately.

This algorithm can be optimized by ordering the packets by their size, from the biggest to smallest, so after putting the first there for biggest packet into the bin we could have the smallest portion of space as our next been therefore more space would be saved.

Although in the main algorithm there is no limitation as the weight for pallets and maximum payload for each pallet it could be implemented in such a way.

Tabu search This algorithm works based on the computed penalties for each bin in the space, so with the calculation of the penalty for each package depend on

how well its fitted to the given space, and find the packet with the minimum penalty for every space it can get the decision of putting the package in that space.

As the main point of this algorithm is based on the given penalties to every individual bins and package to fit trough them it seems to be easier to implement this algorithm based on real life limitation for shipment.

The algorithm suggested in this project is a combination of these two algorithm with some changes in order to make it more suitable to real life conditions.

1.2 Path optimization

Using bin packing algorithm, path optimizer is making sure that each carrier has been loaded up to it's maximum capacity. this will reduce the number of half empty carriers and leads to save the time and cost caused by them.

The programming language chosen for this task is C++. there are two main reason for this, first to ease of use this application should have been design despite the different type of operation system having 3D engine and complex user interface would leave us to choose between C++ and Java, and as for java virtual machine and much higher process time, C++ has been chosen.

Qt is an open source and cross platform library for C++ in order to designing complex GUI and managed data structure such as lists it also has the ability to integrate with openGL, this project has been implemented using Qt, at the beginning of the project 3D engine for showing the result of planning algorithm was Coin3D, a OpenGl based complex 3D engine and scene manager, but due to the complexity in compiling the following library in different platform such as Windows, Mac OS and Linux it's been replaced by OpenGL itself which limits the operation on 3D viewer up to some level.

For ease of use and better understanding of changes in source code a private repository in GitHub has been opened to submit any possible issues or expanding plans.

Implementation 2

This chapter is an explanation about the source code of the Cargo Loading application, first there will be some explanation about the data structure and the choice of use of any specific structure then there will be explanation about different algorithm that has been used and the time all the limitation that has been designed in the core algorithm to make this application more real life like and easier to use.

2.1 Code Standard

To increase the code readability some grand rule has been put through as following:

- all the classes are starting with capital `C` ex. `CClass`
- object that are member and not belonging to a local function are staring with `m_` ex. `m_object`
- pointers also have `p` before their name ex. `pPointer` , `m_pPointer`
- boolean are starting with `b` ex. `bBool` , `m_bBool`
- names starts with capital letter ex. `Name`
- in case of multi words names each word start with capital ex. MultiWord, m_MultiWord , m_pMultiWord
- before the name of the function there are `pragma` to make it easier to collaps the section inside.
- name of the function with the boolean result are starting with `is`, ex. `isFunction()`

- function with getting the information starting with `get`, ex. `getfuntion()`
- function with setting the information starting with `set`, ex. `setfuntion()`
- main classes related to the core optimizer and algorithm are starting with `__` ex. `__ClassName`
- all the classes has constructor to fill up the predefined values

2.2 Data Structure

This application is using the self implemented data structure instead of the usage of data storage third party applications like data bases the reason would be to eliminate any chance of corruption in the procedure of development as the main aim of this application is the algorithm behind it this would help to have better estimation on time level of the main core and to depart any external process time.

For any object in that has been used special class with a name related to that object has been designed, to make the code more readable some standards has been follow in the next section this standards will be discussed. after which there will be explanation about `rows.h` file which is responsible about the majority of the main classes that has been used over the project and algorithm.

2.2.1 `rows.h` header file

This file is in charge of most of the classes that has been used for the main purpose of the optimization and the running of the algorithm there are 9 different classes that has been design for this purpose.

2.2.1.1 `__BoxRow` class

`__BoxRow` has been used to store the main cargo that has been got from the user in cargo tab or has been restored from the `data.dat` saved file. this file has the following options:

- `id`: automatically generated to identify each individual cargo by a unique number the number is an integer.
- `Desc`: user define contains the name or the description of the specific cargo.
- `code`: variable to help to show id in different windows.
- `NodeId`: variable to keep the id of the related node.
- `PaleteSizeD`: Cargo size depth.

- `PaleteSizeH`: Cargo size width.
- `PaleteSizeW`: Cargo size height.
- `PaleteWeight`: Weight of the Cargo.
- `PaleteWeight`: to specify which sides is allowed to rotate the cargo if put as `1 2 3 4 5 6` means the core is allowed to put the cargo in every face and rotate in any direction. if the cargo is only allowed to stand in on face this can be specified like `1 4`
- `Color`: color indicator for the cargo.
- `Priority`: if there are any priority on the cargo (to be implemented)
- `MaxLoad`: the maximum weight that the cargo can carry on top.
- `BoxCounter`; helper variable to auto generate the id and have the counter for the cargoes.

there are two function in this class,`void SaveToBuffer(CBuffer* pBuffer)` and `void ReadFromBuffer(CBuffer* pBuffer)`, to save and load the information from `date.dat` file to prevent the loose of the data on closing the application.

2.2.1.2 `__BoxInContainer` class

`__BoxInContainer` has been used to store the computed data after the run of the algorithm like optimization, it help to store the information for calculated location and in case of any rotation changes in width, height or depth plus their 3D location inside the carrier.

- `w`: calculated width of the cargo(in case of rotation is different to main node width).
- `h`: calculated height of the cargo(in case of rotation is different to main node height).
- `d`: calculated depth of the cargo(in case of rotation is different to main node depth).
- `x`: X location of the box in the carrier.
- `y`: Y location of the box in the carrier.
- `z`: Z location of the box in the carrier.
- `NodeId`: parent node ID;
- `ContainerID`: id of the carrier that has been placed in.

2.2.1.3 `__Provider` class

`__Provider` has been used to store the information of the places the cargo has to been collected from.

- `id`: automatically generated to identify each individual provider by a unique number the number is an integer.
- `name`: name identifier of the provider.
- `Address`: store address of the provider. this variable is not being used inside the algorithm and is just to notify the user.
- `phi`: phi location of the provider optimization of the path.
- `landa`: landa location of the provider optimization of the path.
- `ProviderCounter`: helper variable to auto generate the id and have the counter for the providers.

2.2.1.4 `__CArea` class

`__CArea` has been design to store the free spaces in carrier in order to place the cargoes inside.

- `m_W`: with of the space.
- `m_H`: height of the space.
- `m_X`: X position of the space.
- `m_Y`: Y position of the space.
- `m_Z`: Z position of the space.
- `m_FloorZ`: Z position of the space if it is belong to the bottom of the carrier (for optimization purposes)
- `m_TopZ`: Z position of the space if it is belong to the top of the carrier (for optimization purposes)
- `m_MaxLoad`: indicator to specify how much weight is allowed into that certain space.

2.2.1.5 `__Carrier` class

`__Carrier` has been used to store information about different carrier that can be user to put the cargoes on.

- `id`: automatically generated to identify each individual carrier by a unique number the number is an integer.

- `name`: name of the carrier.
- `Depth`: Depth of the carrier.
- `Width`: Width of the carrier.
- `Height`: Height of the carrier.
- `MaxLoad`: maximum weight that any carrier can be loaded.
- `CostPerKm`: cost of the carrier per kilo meter to use for the optimization of the costs.
- `VahicleType`: type of the carrier.
- `m_FloorAreas`: free spaces at the bottom of the carrier to load the cargoes.
- `m_TopAreas`: free spaces NOT at the bottom of the carrier to load the cargoes.
- `CarrierCounter`: helper variable to auto generate the id and have the counter for the carriers.
- `m_BoxList`: list of the boxed placed inside the carrier.

2.2.1.6 `__Pallete` class

`__Pallete` has been designed to store the relation and priority of the cargoes(to be implemented).

- `id`: automatically generated to identify each individual pallet relation by a unique number the number is an integer.
- `name`: name of the relation pattern.
- `bIsPutOver`: list of relationships in boolean.
- `ProviderCounter`: helper variable to auto generate the id and have the counter for the pallet relations.

2.2.1.7 `__Order` class

`__Order` has been design to store the user's order of any cargoes from any provider the number and the time that item is needed.

- `id`: automatically generated to identify each individual order by a unique number the number is an integer.
- `BoxID`: id of the cargo that has been ordered.
- `count`: number of the cargo that has been ordered.
- `minTime`: the minimum time of the limitation that the requested cargo is needed.

- `mamTime`: the maximum time of the limitation that the requested cargo is needed.
- `distFromCenter`: this variable has been design to store the calculated distance that the provider has with the main point.
- 'ProviderID: id of the provider that has been ordered.
- `Provider`: pointer to the provider object.
- `OrderCounter`: helper variable to auto generate the id and have the counter for the order.

2.2.1.8 `Node` class

this class contains information about each node inside a path that carrier have to visit in order to collect or deliver the goods and cargoes.

- `id`: unique identifier to recognize each individual node.
- `m_Dist`: distance between the main location and node location that carrier has to pass.
- `m_phi`: phi location of the node.
- `m_landa`: landa locatio of the node.
- `m_Order`: pointer to order list.
- `NodeCounter`: helper variable to auto generate the id and have the counter for the node.

2.2.1.9 `Path` class

This class is for store all the information related to each path that carrier have to take plus their node inside it.

- `id`: unique identifier to recognize each individual path.
- `m_Node`: a list of all the nodes inside the certain path;
- `m_Carrier`: the carrier for this path which contains all the location of the box inside it.
- `m_minTime`: minimum limitation that the certain carrier has to leave.
- `m_maxTime`: maximum limitation that the certain carrier has to leave.
- `m_Cost`: cost of this certain path.
- `m_PenaltyCost`: penalty cost for empty spaces inside the carrier to compute the most optimized solution.

- `PathCounter`: helper variable to auto generate the id and have the counter for the path.