SMART CONTAINER LOADING

by

OMAR CHEHAYEB MAKAREM

B.S., Computer Science, American University of Beirut, 2008

Project submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Department of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

May 2008

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

Project Approval Form (Annex V)

Student Name:Omar Chehayeb Makarem I.D. #: 200400124

Project Title : Smart Container Loading

Program : Computer Science

Division/Dept : Computer Science and Mathematics

School : School of Arts and Sciences

Approved by: Ramzi A. Haraty

Project Advisor: [redacted]

Member : Abdul Nasser Kassar [redacted]

Date May 20, 2008

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Omar Chehayeb Makarem

Signature: 

Date: May 5, 2008

I grant to the LEBANESE AMERCIAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.

Acknowledgment

I would like to express my extreme appreciation to my advisor Dr. Ramzi Haraty for his guidance and encouragement throughout my project work. I would also like to thank Dr. Abdul Nasser Kassar for being on my project committee.

I want to expressly deepest gratitude for my parents who kept supporting and motivating me to complete the project. This work is theirs as much as it is mine.

To my beloved three sisters, Suzanne, Tamara and Dalia, your love, care and affection will always be remembered. May you all be proud of me as much as I'm proud of you.

To all my friends who eased the burden of the work, I thank you. You are all a source of inspiration and admiration. A special thanks goes to my former supervisor and mentor at work, Mr. Raed Gharzeddine, who suggested the topic.

Abstract

The need to move various items from one location to another has become a primary need of most companies. Items are usually boxed and arranged inside containers to be transported from one location to the other. Most companies pay shipments by the number of containers being transported; therefore, it is in the interest of the company to minimize that number. Furthermore, some items might have properties, which force special constraints on how and where they are stored inside the container. In this work, we propose an agent based solution for maximizing the use of a single container; thus, reducing the transportation cost for the companies having their items transported, as well as increasing transportation performance for transportation companies. Separate agents are used for packing, tracking, loading and unloading items. Also, artificial intelligence is provided through the use of the PROLOG programming language to specify rules and constraints on items loaded as well as on the container itself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Movement of items of any type, shape, size and weight has become increasingly common among most companies; industrial and commercial, as well as among regular individuals for both business and personal reasons (Birgin, Martinez & Ronconi, 2005). Items being transported can range from small items of irregular shapes having negligible weight, to large heavy items with specific shapes. Companies transport items for business reason. It is common among construction companies to transport equipment used in one site, to be able to use them on another site, instead of buying new material for each site. It is even more common among production companies to ship their products from their factories to various locations where their consumers reside. Regular individuals, as well as companies, may need to transport their possessions from location to another, most notably when moving from a residence location to another. Their possessions will include furniture, electronics and many other personal belongings.

Depending on the type of the item being transported, specific properties might be associated with the items. Some items can be marked as fragile, these items will need special care when being placed, and cannot be placed anywhere when being transported. Some items might be of more value than others, and will need to be handed priority when being transported. Each item however belongs to a specific customer. These items must reach the destination specified by the customer. When having his items transported, the customer will expect his/her items to arrive at a

specified date, to the specified destination. Therefore, items of the same customer are usually transported together.

Shipping companies have to deal with the many types, shapes, sizes and weights of these items. These companies are responsible for shipping the items from one location to another. They might ship items by land through vehicles, such as pick up trucks, or by sea through ships. In either case items that are to be transported will be placed inside boxes. All items are boxed in order to provide a single shape to deal with, ease of handling and protection when transporting (Birgin & al, 2005). Providing a single shape simplifies the problem by reducing the infinitely many shapes that items can take into a single shape: a box. Furthermore, some items might be too small to consider them by themselves. That is why we group them with other items inside the same box. Boxes are also easier to move around and protect their content from damage (Birgin & al, 2005). Some companies may have only limited amount of box types to be used to further simplify the problem. Even when boxed, these items cannot be transported alone. The shipping companies will usually group boxes in containers for each separate shipment. These containers will serve as the smallest units of transportation. Shipment companies will ship a number of containers in every shipment.

It is in the benefit of both the transporting company and the customer to keep the number of containers needed to transport this customer's items to a minimum (Birgin & al, 2005). Since the customer pays the transportation company according to the number of shipments taken to transport his/her items, by keeping this number to a minimum, he/she will reduce his transportation cost. The shipping company will also benefit from this by increasing their transportation performance. This will allow them to serve more customers in the same amount of time, increasing their profits.

In order to keep the number of containers needed at a minimum, we are faced with the problem of maximizing the utilization of a single container. In most cases, maximizing utilization refers to maximizing the space of the container occupied by boxes, but, depending on the situation, it might be more interesting to maximize weight, number or other aspects of the boxes inside this container. This problem is commonly known as the Container Loading Problem (Pisinger, 1999; Scheithauer, 1999).

This type of problem has numerous applications, mostly in the packing industry as we have seen, but also in the cutting industry, for optimizing the cutting of wood or metal into smaller peaces (Alvarez-Valdes, Parenno & Tamarit, 2003).

In the following chapter, we discuss the theoretical aspects behind this problem as well as similar computer science problems.

1.1 Container Loading and Related Problems

Container Loading is a classic problem mainly due to its simplicity in defining it and the complexity of finding a solution for it. This problem cannot be solved in time relative to the number of available boxes; and therefore, it is a non-polynomial (NP) hard problem (Scheithauer, 1999). There are a lot of variations of this problem, most commonly the "Bin Packing" (Faroe, Pisinger & Zachariasen, 2003; Martello, Pisinger & Vigo, 2000), "Cylinder Packing" (Birgin & al, 2005) and "Pallet Loading" (Pureza & Morabito, 2006; Lim & Zhang, 2005) problems. By simply modifying the number of dimensions used, shapes of items placed or number of containers to place items in, we end up with a new variation of the problem.

All the variations of this problem can be summed into a single problem called the "Packing Problem". The Packing Problem is simply an application of the more general "Multidimensional 0-1 Knapsack Problem" ("Knapsack problem", 2007). The

Knapsack Problem, which itself has a number of variations, is, in its simplest form, a maximization problem which consists of maximizing the value of the items that can fit in the knapsack (Vasquez & Hao, 2001). Being multidimensional means that the problem is subject to more than one constraint, since it is a 0-1 problem, items available can be either included in the knapsack or left behind, one can not however take part of that item (Vasquez & Hao, 2001). The problem can be stated more formally as:

> Given a set of **n** objects where each has a corresponding profit, and a knapsack, pack the list of objects that can fit in the knapsack such that the profit sum of included objects is maximized (Vasquez & Hao, 2001; Pisinger, 1994).
>
> The problem is also defined mathematically in "Knapsack problem" (2007) as follows:

$$\text{Maximize} \sum_{j=1}^{n} c_j \cdot x_j$$

$$\text{Subject to} \sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i, \text{ for } i = 1..m$$

$$x_j \in \{0,1\}, 1 \leq j \leq n$$

> where **n** is the number of items available to be packed, **m** is the number of constraints that the solution must adhere to, the value of a specific item **j** is represented by c_j, x_j indicates if item **j** is included in the result or not, if the item is included the value is 1 while if it was excluded, the value is 0, b_i is the maximum value for constraint **i** that the solution can have. Finally, $a_{i,j}$ is an entry in a constraint matrix which contains the value of each item **j** on each constraint **i**. (p. 1)

For the container loading problem, we are maximizing the value of the boxes loaded in the container (Lim & Zhang, 2005). This value may vary depending on the aspect we want to maximize. Sometimes, we need to maximize the capacity of the

container used, so we try to maximize the volume of the boxes chosen (Lim & Zhang, 2005). In other cases we might want to maximize the value of the weight inside the container, so we focus on the weight of the boxes inside the container (Bischoff, 2003). In some other cases, we might even need to maximize the number of the boxes to be packed. So this value will depend on the logic we intend to follow.

The container is subject to many constraints (Bischoff, 2003). Most notably, the length of the container on each of the three dimensions must not be exceeded by the largest sum of adjacent boxes' length on that dimension (Birgin & al. 2005). Another recurring constraint is that of the container's weight: the sum of the weights of the items inside the container must not exceed the weight capacity of the container (Bischoff, 2003). Many more complex constraints can be added to the container, such as load balance of boxes inside the container or constraint on location of objects inside of it.

The container is not the only subject to constraints. Some boxes may be constraint by the weight they can carry inside them or on top of them. Some boxes may not even allow items to be placed on top of it at all (Bischoff, 2003). In other cases, the direction of the faces of the box might even be constrained.

This problem cannot be solved since it is a NP-Hard problem. In the next section we discuss how such problems are usually handled.

1.2 Methods Solving the Problem

Being a NP-Hard problem, one cannot find the optimal solution for this problem in polynomial time; i.e, time relevant the number of boxes we have as input (Martello& al. 2000). In order to deal with such problems, people usually revert to Heuristics (Lim & Zhang, 2005). Heuristics provide near optimal solution for large number of instances (Pisinger, 2002). A heuristic is an algorithm that does not get the

optimal solution for the problem; it gets a solution close to the optimal one in good and reasonable running times. In many problems, including the container loading problem, it is more practical to use heuristics to find solutions almost as good as the optimal one in an acceptable amount of time, then to wait indefinitely till we find the optimal solution for a problem (Pisinger, 2002).

1.3 Organization of the Report

In the next chapter, a review of literature related to container loading and the use of heuristics is made, showing how others used heuristics and other techniques to solve this problem. Chapter 3 contains the proposed solution for this problem. Chapter 4 discusses the results of this solution and chapter 5 serves as a conclusion.

Chapter 2

Literature Review

As stated before, container loading problems can be solved in acceptable times only through the use of heuristics. All the work related to this topic in the literature review consists of proposing a new heuristic or choosing an existing one and extending it, applying this heuristic, and showing its performance over the problem.

The simplest heuristic to use would be the greedy approach (Lim & Zhang, 2005). In this approach, boxes are chosen in a greedy manner, starting with the most desired box and trying to fit it in the container, then going through the remaining boxes following the same approach until all boxes are chosen or the container is full. In practice, since the objective is to maximize the value of boxes in the container, the most desired box to pack would be the one with the highest value. If not packed early, these boxes will be awkward to pack at later stages. This approach is very simple to implement and very fast in practice. However, it hardly yields near optimal or even acceptable solutions.

In their work, A. Lim and X. Zhang (2005) extended the simple greedy approach by suggesting some "trouble-making" (Lim & Zhang, 2005, p.914) elements: boxes that are awkward to pack. These elements will force the use of dynamic prioritization between iterations in the algorithm. The "trouble making" aspect is represented through a priority factor assigned to each box type, this factor is dynamically changed to increase or decrease the priority of a specific box. The greedy approach is still followed, but here the boxes with the highest priority, not the ones with the most value, are being chosen.

Using this approach the most "trouble making" boxes are always placed at the bottom of the container. The idea is to get rid of these difficult to place boxes at the beginning, in order to deal later with easier boxes to place. The issue in this approach is that it does not build a layer on which other layers or boxes can be placed, rendering the placement of boxes on higher levels difficult.

Common approaches to solve this problem include the wall-building approach (Pisinger, 2002). In this approach the container is filled with a number of layers across the depth of the container. The same approach could be used to fill the height of the container instead of the depth and is called the stack building approach (Pisinger, 2002). The depth, or height, of a layer in either approach must be well chosen. Usually, boxes are sorted based on their smallest dimension, the box with the largest smallest dimension is then chosen, and the depth is chosen equal to the largest side of that box. These boxes will be difficult to accommodate later, so are chosen first. This wall building method also fallows the greedy approach and is bound to the same weaknesses.

Pisinger (2002) extended this greedy approach of wall building to include back tracking, by using a tree search heuristic. The tree search algorithm is used to find the layer depths that will provide the best overall filling. Even though not all possible branches are studied, due to computational complexities, the heuristic has some good results.

The wall building approach, whether using the greedy or the tree search heuristic, is designed to fill containers with boxes based on their volume. This approach is not well suited to fill the container based on other aspects of the boxes, such as their weight or their priority.

Linear programming is another common method used in heuristics for the container loading problem (Schithauer, 1999). Linear programming is used to find the best outcome; maximum or minimum, for a linear function given equality and inequality constraints ("Linear programming", 2007). Linear programming is mostly used for optimizations in the operation research, microeconomics and business management ("Linear programming", 2007).

In his work, Scheithauer (1999) was able to find tighter bounds for the container loading problem through the use of linear programming relaxation. Scheithauer argued that the three dimensional packing pattern of the container can be described by smaller spatial dimensions. Through the use of a single dimension: length; the packing pattern of the container can be described through a set of patterns on this dimension, called the bar-patterns. The container is thus partitioned into bars having 1×1 cross-section (1 unit in height and length). The real position of the bar is not restricted and is said to be "relaxed" (Scheithauer, 1999, p.202). These bars contain parts of the packed boxes and can be characterized by integer vectors and are subject to constraints. The problem becomes a linear programming problem by adding the objective function to be minimized: the function showing value of packed boxes. To achieve a tighter bound for the container loading problem, the linear programming problem must be solved. Similarly, Scheithauer was also able to describe the packing pattern using a set of two dimensional patterns. This method also works for the multiple containers (Scheithauer, 1999).

The drawback of this method is that it only considers orthogonal packing of the boxes, while in reality; the box has six orientations to be considered. Boxes with same sizes and shapes but with different orientations are considered boxes of different types.

As opposed to the previous methods; which deal with boxes themselves to find the solutions, researchers can resort to search for the best solution in the solution space. This can be achieved through the use of meta-heuristics ("Metaheuristics", 2007). The most widely used meta-heuristics include local search, branch and bound, and genetic algorithms.

Local search iteratively searches solutions moving from a solution to one of its neighbors while modifying the neighborhood structure as the search progresses, until some criterion has been satisfied ("Local search", 2007). Tabu search is a special type of local search because it uses a Tabu list, containing list of solutions visited in the recent past ("Tabu search", 2007). The solutions in the Tabu list are excluded from the neighborhood of the current solution.

Branch and bound also searches the complete solution space. It uses branching to divide search space into subspaces recursively, and bounding to find the upper and lower bounds of the studied branch ("Branch and bound", 2007).

Genetic algorithms alter the pool of solutions by combining or mutating existing solutions, better solutions survive while solutions of lower quality are discarded. The process is repeated until an acceptable solution is found ("Tabu search", 2007). Bischoff (2003) used genetic algorithms to add the load baring aspect to the problem.

In their work, Pureza and Morabito (2006) used Tabu search heuristic to solve the pallet loading problem. The method proposed generates an initial solution using block heuristics, then apply block expansion moves. In block heuristics, block patterns are formed out of smaller boxes arranged in the same orientation. In block expansion moves, neighboring blocks can be decreased in size, divided into one or more blocks, or grow in size in order to preserve the solution feasibility.

Martello, Pisinger and Viro, (2000) developed an exact algorithm with a continuous lower bound to fill a single container. Their algorithm starts by solving the problem in two dimensions, then uses a branch and bound algorithm to develop it into a container loading problem solution: All the boxes placed on the lowest level are then used as bases for sub-containers ranging up to the ceiling of the container, and the problem is repeated for this sub container. This algorithm can also be applied to the bin packing problem. Although this algorithm is easy to implement and has a lower bound, it does not perform as good as other studied algorithms.

Faroe, Pisinger and Zachariasen, (2003) solved the bin packing problem. Their solution first solves the problem using the greedy approach then iteratively decreases the number of bins using a guided local search algorithm. This algorithm also restricts the orientation of the boxes.

In this chapter, some of the most common and popular techniques and heuristics used to solve the container loading problem have been covered. In the next chapter, we will propose a method to solve this problem through the use of agents and by specifying logic for loading the container through a PROLOG file.

Chapter 3

The Solution

In order to solve the container loading problem, rules and constraints to abide by need to be set. The problem is also subject to some assumptions that must be stated. Assumptions, rules and constraints set the frame for the solution. The solution is based on four agents: one for each of the packing, tracking, loading and unloading activities. While packing, items are placed inside boxes and the constraints and rules are set. Tracking locates items in boxes, boxes in the container and the location of the container itself. The loading activity involves loading boxes in the container. The logic of the loading activity can be modified by specifying the logic to follow through the use of PROLOG. Unloading deals with removing all the boxes from the container, finding the fastest way to a box in the container and the lessons learned from the loading activity.

3.1 Assumptions, Rules and Constraints

By definition, the problem only deals with boxes. It is assumed that all items to be transported are packed in boxes that can fit them. Smaller items can be grouped and packed in a single box. This means that a box can contain more than a single item. These assumptions simplify the problem by eliminating the need for smaller boxes for the small items or having a larger empty box.

Once a box has been placed in the container it is assumed that it will stay still in that location for the duration of the shipment. In practice, this can be easily achieved through the use of ropes or by filling the empty space of the container with polystyrene for lighter items or wood and rubber compartments for heavier items.

Items placed inside the boxes are also assumed to stay still during the shipment. The idea is preserve the items that are being transported. This can also be easily achieved through choosing the right box sizes for the items and holding the item in place by filling the empty parts of the box with polystyrene or rubber, if necessary.

Boxes can be made of different material and sizes. Boxes made of the same material and sizes on all dimensions are considered of the same type. It is assumed all boxes of the same type have the same characteristics such as the weight that they can carry in or on top of them.

When boxes are packed, it is assumed that a box can not hang in the air. In practice, it is very difficult or even impossible to place a box without supporting boxes under it. Each box must be on the floor of the container or on top of another box.

All boxes must be packed orthogonally. This rule forces the faces of the boxes to be always parallel with those of the container. In practice, if the box is placed in an inclined position it is highly probable that it will slide and fall and damage its content or the content of other boxes. This last rule takes care of such situations.

Because boxes are placed orthogonally, a box only has six possible orientations. In each orientation the different sides of the box are parallel to different sides of the container. However, some boxes may require a specific face of the box to be facing a specified direction. This is very common in practice; boxes are marked with "This side up" tags to specify orientation.

When boxes are placed in the container they can be placed next to each other or on top of each other but without overlapping. If boxes overlap, the box itself, and possibly its content, will be damaged.

Each box has a limit weight that it can carry on top of it. This weight is assumed to be maximum weight the box can carry without being damaged or deformed; otherwise, the boxes will overlap. All boxes of the same type are supposed to have the same limit. Some boxes might be so fragile that they require that no other boxes be placed on top of them.

Box types also have a limit on the weight they can carry inside the boxes. In practice, if the item inside the box is too heavy, even if it fits in the box, it will damage the bottom of the box. Therefore, items must be place only in boxes that can carry them.

In order to find where items and boxes are in the container, every box and item is tagged with a unique serial number. The serial number is used to identify the location as well as the owner of the items.

The location of the box in the container may need to be restricted. In practice, some items might need to be placed on the floor or next to a side of the container for support. Sometimes a box with a very high priority might be restricted to stay close to the entrance of the container for easy access.

The container itself may have restrictions on the location of boxes. In the case where we have boxes belonging to different customers, it might be required to put the boxes of the same customer near each other. Here, the container will place a constraint restricting each customer's boxes to a specific location.

The container may do load balancing on placed boxes. If the container is a truck, it is required that the center of gravity of the container be close to its middle, otherwise the truck risks flipping over.

All the assumptions just stated are assumed when the solving the problem. All specified rules and constraints must also be satisfied. The assumptions, rules and

constraints constitute the environment in which work is done to solve the problem. Next, in the environment just set up, the use of agents is discussed.

3.2 Agents

As stated earlier, the container loading problem is used in practice to simplify transportation of items from one location to another. Transporting an item in a container involves several steps and underlying processes. First, it involves packing the item into a box so it can be placed into the container. Here, we refer back to the assumption that all items transported must be within boxes. Second, the box must be loaded into the container along with other boxes, in a way that optimizes the use of the container. Finally, the box needs to be retrieved from the container when it reaches its destination. During all the steps, location of the box and the container must be known; therefore, they should be tracked.

In order to automate the transportation of the item, we deal with agents responsible for the processes involved. An agent is software that assists, or acts on behalf of, the user in performing repetitive computer related tasks (Haag, Cummings & Phillips, 2007). Agents apply reasoning capabilities to reach a conclusion. Agents are a form of artificial intelligence where machines imitate human thinking. They are used to replace the tedious and repetitive task that a human has to do every time a set of items needs to be transported. When given the logical thinking to follow, the agent will come up with the same result a human would in much less time.

For each of the four processes involved in the transportation of the item, an agent has been created. The agents share data about the boxes, containers, items and other information through a database that they all use. The database stores information about the box types, boxes, the container itself and the items being transported. The database identifies which items are in which boxes; it also specifies

different locations. Furthermore, the destination of the box must be known to know in which container it should be loaded. Description is again optional only for the boxes that need additional information.

After identifying all the items and boxes to be used, the packing agent notes into which box every item is being placed. This will allow the tracking agent to identify what every box contains and in which box every item is contained.

Next, the container in which the boxes will be placed must be specified. A container has a name, description, destination, height, length, width, and weight capacity. In practice, the container could be the back of a truck or a storage room in a ship among others. The container is identified by its name. Additional information of the container can be provided through its description. The dimensions of the container can be easily measured, while its weight capacity is usually estimated.

Constraints and rules can be added to the problem through the use of properties. Properties can be applied to items, boxes, box types and containers. Properties have a name, a description and a restriction. The name identifies the property. Additional information can be provided for the property through the description. The restriction includes the rule or constraint that restricts the loading operation.

All properties to be used are identified by the packing agent. The agent also assigns these properties to the container, box, box type or item. These properties will be used by the loading agent to enforce rules and constraints on the loading operations.

3.2.2 Loading Agent

The loading agent is responsible for loading the boxes in the container. This agent will take the list of available boxes in the database and try to load them into the

container. The loading logic varies through two Prolog files, one for the Prolog facts and the other for Prolog queries.

Prolog is a programming language used for artificial intelligence programming (Cook, 2004). Like all other artificial intelligence software, Prolog attempts to imitate human behavior by attempting to understand and analyze information and knowledge. For Prolog, there are two main types of knowledge: facts and reasoning procedures. Facts are information that is known to be true, while reasoning procedures, or queries, follow reasoning of facts (Cook, 2004).

The .Net framework is used to create the agents. In order to read Prolog files using this framework, a library called P#, created specifically to bridge between .Net and Prolog, is used.

The loading agent works by retrieving the loading logic from the Prolog facts and queries, then working its way through the boxes available in the system and tries to fill them in the most optimal way. To do so, the P# is first used to load the facts file, and the queries are then executed against these facts. The answers to these questions are reported in an output file. The agent will read the results from the output file, and according to these results it changes its logic and behavior.

According to the logic specified in the Prolog file, the agent will sort boxes by specified criteria. The agent will try to insert these boxes according to their order into the container.

The loading is achieved through the use of sub-containers. When the problem starts, there is only one sub-container, which is the container itself. Every time a box is inserted in a sub-container, it creates three sub-containers formed by the volume between a box's faces and sub-container's faces parallel to them. If a sub-container is too small to fit any box, it is discarded. Sometimes, when a box is placed, it intersects

existing sub-containers and thus reduces their size. The program continues until the smallest box is larger than the largest sub-container and no more boxes can be placed in the container. The resulting load is the solution to the problem.

Loads can be temporarily stored and continued later. This happens when a list of boxes is inserted and a load is executed on these boxes, then more boxes are added to the system. In this case, the user has the option to continue loading the remaining unloaded boxes in the container while keeping already loaded boxes in their places, or to remove the loaded boxes from the container and repeat the load with all the boxes in the system.

3.2.3 Unloading Agent

The unloading agent is primarily used to unload boxes from the container, and items from the boxes. Just like the packing agent is an observer of packing operations, the unloading agent is an observer of all unloading operations. Every time a box or item is unloaded, the agent notes the change in the database.

The unloading agent uses features from the tracking agent to get the list of boxes blocking a specific box. This agent determines the easiest and fastest way to get to that specific box, and the items inside it, by determining list of boxes that need to be removed first.

While unloading boxes, the agent can add lessons learned in form of Prolog facts and queries. The packing agent adds constraints and rules imposed by the user through properties. The unloading agent on the other hand changes the logic of the loading activity for future loads by adding facts and queries about the last load. Facts might include lessons related to the location where the box should be placed, such as placing wooden plates on the top of other objects and not on the floor. The queries that might be added will change the logic of the loading operation. Queries are

questions that will be asked and checked against current facts, such as "should wooden plates be placed on top of other objects?"

The agent is able to clear the database. This feature is used when a new experiment is to be performed. In practice, the old data and information is usually backed up, and then the database is cleared to allow the new experiment to be performed.

The agent is also able to delete all types of objects in the system. This feature is used when one box, box type, item or container were incorrectly added, or when we want to perform the experiment without the specified object.

This agent is also able to remove constraints and rules applied to the problem by removing properties associated with boxes, box types, containers and items.

3.2.4 Tracking Agent

At any point, the tracking agent is responsible for identifying the boxes, box types, items, containers and properties in the system. Although the information about these components is inserted in the packing agent, the tracking agent is responsible for retrieving this information. All the retrievals performed by the tracking agent are from the common database, making this agent the search engine of the system.

Another peace of information inserted at the packing stage and retrieved at this stage is the items contained in a box. Given a specific box, the tracking agent will retrieve all the items contained in that box. On the other hand, the agent is able to find the box containing a specified item. The agent is also able to identify all the boxes, and thus, items contained in the container. Furthermore, given a specific box, box type, item or container the agent can find associated properties. Conversely, given a property, it can find all boxes, box types, items and containers that are associated to this property.

This agent also keeps track of the location of items and boxes. At any point, given a specific box or item, the agent is able to find out if it is loaded in a container, and if it is loaded, the agent can specify its exact location. Conversely, given a specific location inside the container, the tracking agent is able to find the box at that location. These features allow the unloading agent to find the list of boxes that need to be unloaded in order to reach a specific box.

Other activities that agent can do include finding the number of boxes of a specific type. The agent is also able to identify unpacked items and unloaded boxes in the system.

3.3 Graphical User Interface

In this solution for the container loading problem, a graphical representation of the loaded boxes is given. This representation is achieved through the use of Microsoft's .Net framework and an OpenGL library for .Net.

The .Net framework is a windows component that supports building and running applications (Cook, 2004). The advantage of .Net is that is simplifies development, deployment and integration with a wide variety of programming languages. In this work, C# is used as the coding language to write the agents.

C# will communicate with an OpenGL library created specifically for .Net. "OpenGL is a software interface to graphics hardware" (Neider, Davis & Woo, 1997). OpenGL is responsible for sending information to the hardware that will graphically draw the solution.

The graphical user interface lists all the containers, boxes, box types, items and properties in the system. By selecting a specific object, the system will display detailed information about that object (As shown in Figure 1).

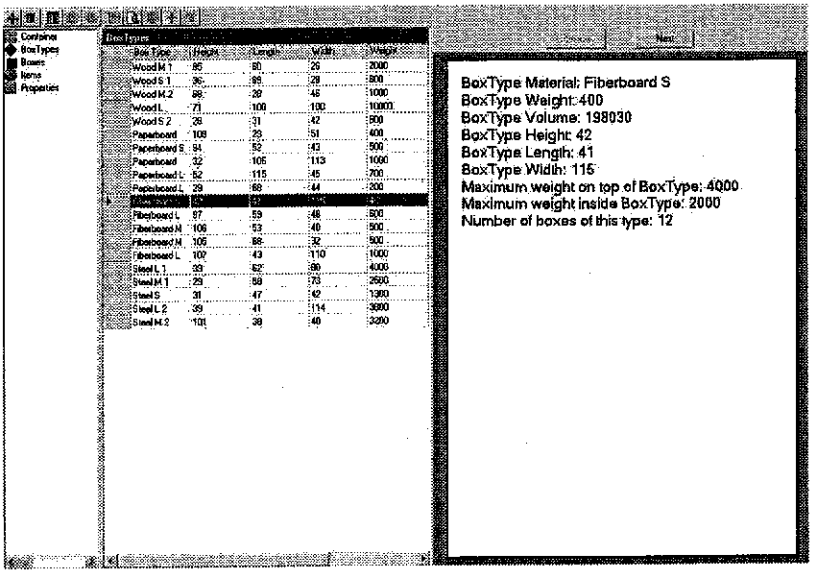

Figure 1 Information about objects

The interface also provides controls to add new objects (As shown in Figure 2), delete existing objects or associate other objects with properties. It also allows the user to change the Prolog facts and queries to be used in the loading process (As shown in Figure 3).

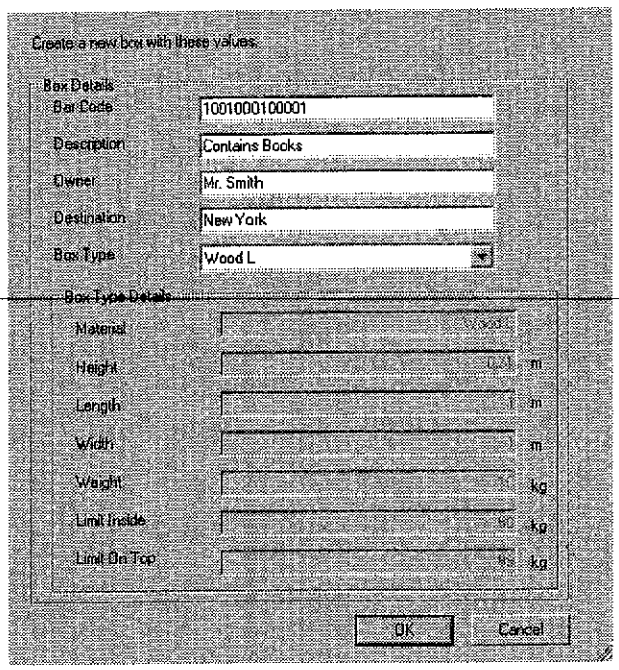

Figure 2 Inserting new objects

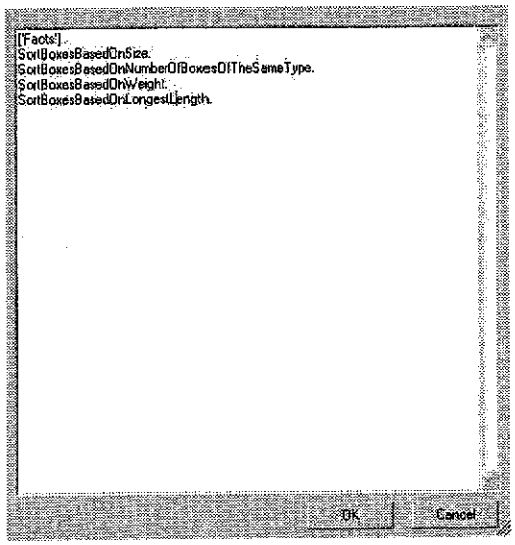

Figure 3 Prolog queries

At any point, the user can empty the container and then refill it using the changed Prolog logic. This can be achieved through the interface by the assigned buttons for these actions.

The graphical user interface will load information about the container and the boxes inside this container. The user interface will draw only the frame of the container, in order to see through it. The interface will also draw solid cubes for each box with randomly chosen distinct colors for each box in order to identify them easily. The interface allows for rotation around the container to view it from different angles. It also allows for zooming in and out of the container for a better view. (As shown in Figure 4)

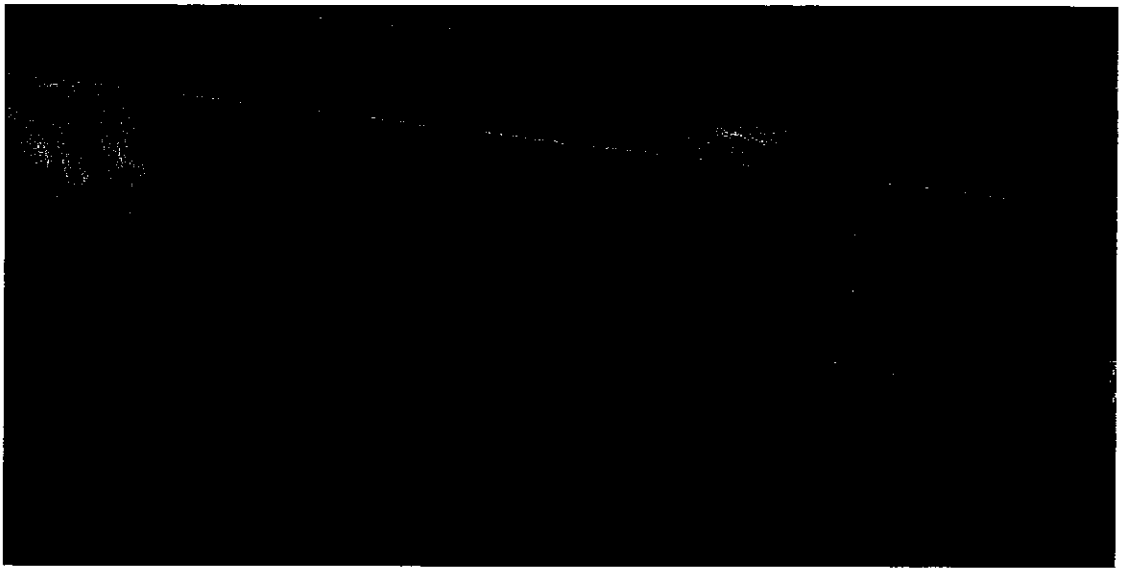

Figure 4 Container graphical view

When user is trying to find the shortest way to reach a box, the interface will only draw the frame of the boxes that need to be removed in order to reach that box (As shown in Figure 5). If the box can be directly removed, all the boxes will be drawn normally.

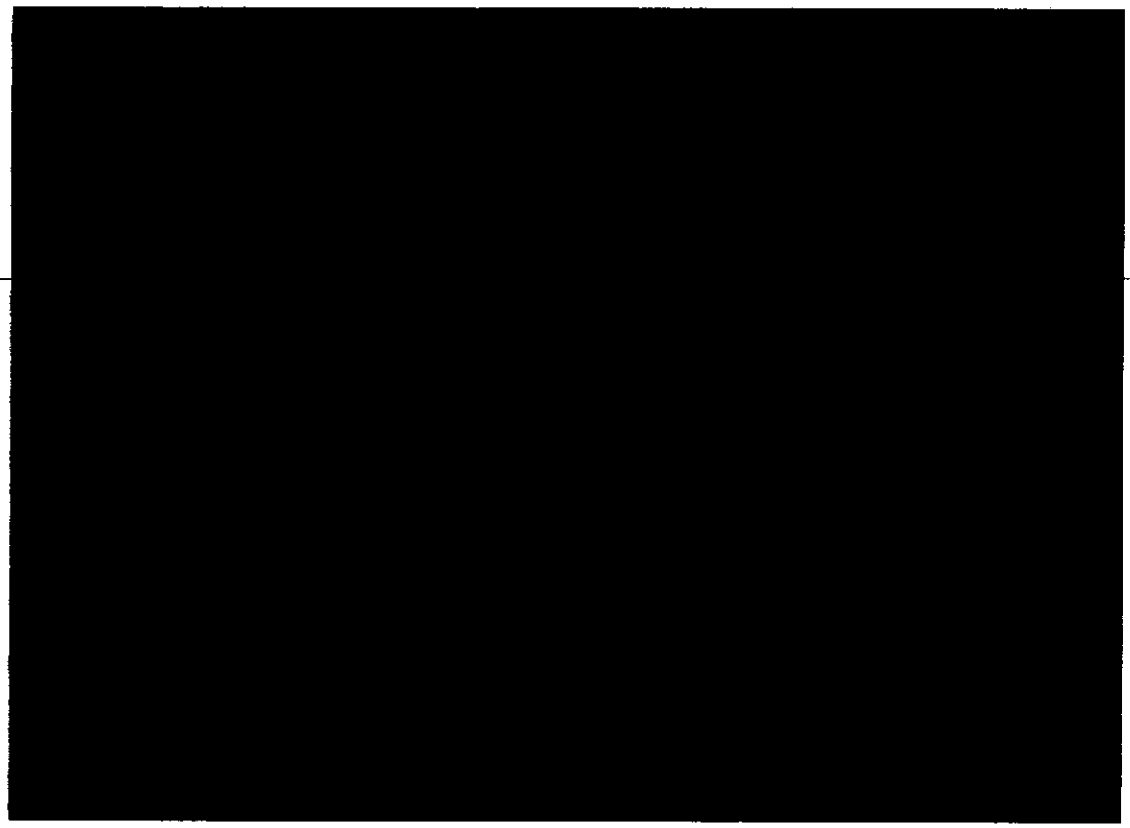

Figure 5 Box inside the container

This interface will be used to test the loading logics that created using the Prolog facts and queries. In the next section, experiments will be performed on the user interface to test the effectiveness of the solution.

Chapter 4

Result and Discussion

In order to test the effectiveness and efficiency of the proposed solution, it will be benchmarked against numerical data taken from Pisinger (2002). That data however does not contain material type or weights for the boxes. To solve this problem, material types were randomly assigned to each box type out of the following materials: paperboard (for carton), wood, fiberboard (for corrugated boxe) and steel. Boxes weights, maximum weight that can be carried inside and on top have been assigned to each box type depending on its material and its volume. The list of boxes used as input is detailed in the table below.

Table 1: Dimensions and properties of input boxes

Box type	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
Width	85	36	68	71	28	109	94	32	52	29	42	97	106	105	107	39	29	31	39	101
Height	60	69	26	100	31	29	52	106	115	68	41	59	53	68	43	62	56	47	41	38
Depth	26	28	46	100	42	51	43	113	45	44	115	48	40	32	110	80	73	42	114	40
Material	w	w	w	w	w	p	p	p	p	p	f	f	f	f	f	s	s	s	s	s
Weight	2	.6	1	10	.5	.4	.5	1	.7	.2	.4	.6	.5	.5	1	4	2.5	1.3	3.8	3.2
In Weight	10	3	5	50	2.5	2	2.5	5	3.5	1	2	3	2.5	2.5	5	20	12.5	6.5	19	16
Top Weight	20	6	10	99	5	4	5	10	7	2	4	6	5	5	10	40	25	13	38	32
Boxes	5	7	6	5	6	8	9	6	13	9	12	8	5	3	3	6	7	10	8	10

The container chosen to fill these boxes in has the same dimensions of the one used in Pisinger's (2002) data. The container has a height and a width of 230 centimeters and a depth of 590 centimeters for a total volume of 31,211,000 centimeters cubed. The 146 boxes have a total volume of 91.8% of the container volume. Pisinger's (2002) algorithms were able to fill up 90.44% of the containers volume with specific ranking rules; however some tests took up to two minutes to come up with the result.

The user interface is used to enter all this data as input to the problem. Behind the scenes, the loading agent is responsible of entering boxes into the database. The container is assigned a default capacity of 1,000 kg. In order to compare the results to Pisinger's outcomes, the boxes will be loaded empty into the container.

In order to try to maximize the utilization of the container, prioritization logic is used. The prioritization logic to use is specified through the PROLOG file. The logic will order boxes based on the specified priority. The loading agent will attempt to load boxes into the container based on this order. Results of prioritization logics based on volume, weight, number of similar boxes available and length will be discussed. For these tests it is assumed that the boxes loaded cannot be rotated and must be placed as defined.

First, the weight maximization of the boxes inside the container is studied. Boxes are ordered based on their weight, with heavier boxes having higher priority. Doing so should increase total boxes weight, filling a higher percentage of the weight that the container can hold. This logic fills 92 boxes in the container which constitute 63% of the volume of the container and 20% of the weight that the container can hold. This finding is rather expected since this logic would sacrifice volume for the sake of weight. Out of the studied logics, this is the logic that maximizes weight utilization of the container. This logic is not very effective when the deviation of weights of the boxes is rather small. On the other hand, it is very useful when boxes are filled with actual items of different weights.

Next, the volume maximization of the boxes inside the container is discussed. Following the greedy approach, boxes are ordered based on their volume in descending order. Boxes with larger volumes will be inserted first into the container. Running this logic will fill 73 boxes into the container. These boxes constitute 66% of

the volume of the container, while their weight is 12% of the weight that the container can hold. The low number of boxes packed is explained by their large volumes, since fewer boxes are needed to fill more space in the container. The volume filled percentage is still rather low and better results can be achieved.

Inspired by the wall building approach, the following logic prioritizes boxes based on the number of boxes of the same type that are available. The idea behind this logic is that boxes of the same type constitute perfect walls, reducing wasted space between them. This test fills 125 boxes in the container which constitute 67% of the volume of the container and 16% of the weight that the container can hold. Even though this logic loads the most boxes into the container, it still does not provide a much better solution then loading based on volume. The reason is that there are usually more small boxes and fewer large boxes. In this case, the smallest boxes were being loaded, increasing amount of loaded boxes but not greatly affecting volume filled.

Finally, the logic that orders boxes based on their length is studied. Here, boxes with the longest length have higher priority. This logic also imitates the wall building approach. Here the walls are defined by the lengths of the boxes. When a box is placed inside the container, it defines a wall segment having for width the length of the box. Since boxes are prioritized by their length, all smaller boxes are candidates for that wall segment. This test fills 101 boxes in the container which constitute 71% of the volume of the container and 15% of the weight that the container can hold. This logic is also similar to that of Pissinger's, except it is only without allowing boxes to rotate and wall segments are determined by box's length. This logic proved to be the most efficient among the studied logics. These findings agree with the reviewed

literature which shows that wall building approaches are more effective than greedy approaches.

Pissinger's (2002) approach can also be implemented in the logic. This can be achieved by adding rotation to boxes and prioritizing based on the largest smallest dimension. Pissinger builds his wall segments based on the smallest dimension of a box, so prioritizing boxes based on their smallest dimension should give similar outcomes to those in his research.

Regarding speed, some of Pissinger's tests took up to two minutes to come up with results. The loading agent came up with results relatively faster. It took the agent less than five seconds to fill the container.

The studied tests are all executed with empty boxes. Adding items to the boxes will increase the constraints on the problem. The resulting outcomes will be constrained by the weight and positioning constraints enforced by the items on the box or on the container.

Chapter 5

Conclusion

In this work, a new way to solve the container loading problem using agents has been studied. Four agents were used to perform the various tasks involved in transporting items: a packing agent, a tracking agent, a loading agent and an unloading agent. These agents communicate with each other and share information and data by using a common database.

A set of assumptions, rules and constraints have been considered and applied to the problem in order to solve it. It is assumed that the problem only deals with boxes that stay still with their content inside the container. Boxes must be packed orthogonally in one of six orientations, without them overlapping other boxes; furthermore, boxes can not be hanging in the air. Restrictions may be applied for a box's location, or the weight it can carry inside or on top of it.

The contribution of this work is in the agents created. A packing agent acts as an observer for the process of filling the boxes and assigns properties to each box or item and to the container itself. A tracking agent is responsible for locating boxes and items at all times.

An unloading agent is used to unload boxes from the container. This last agent is capable of learning lessons when boxes are unloaded. It is also able to find the shortest path to reach a box in a container by identifying required boxes to be removed to reach that box.

A loading agent is responsible for placing boxes in the container in a way that optimizes our required attribute. This agent that can have its loading logic changed

easily through Prolog. This capability allows control over complexity and type of loading to be used.

The proposed solution is rule based in that it allows easy addition of new constraints both on the container and on the box. It also allows changing the logic of filling the container, which allows it to change the complexity of the problem depending on the user's requirements.

A graphical presentation of the problem was provided to visualize the problem and the solution. The open source graphical language, OpenGL, was integrated with Microsoft .Net via a library created specifically for .Net in order to provide the graphical presentation.

The discussed agents, that solve the container loading problem, are highly useful and effective. These agents need to be applied in a real work environment to fully test their abilities and discover their limitations.

Further work can be done to make these agents be used as framework for testing loading logics and visualizing the solutions.

References

Alvarez-Valdez, R. Parreno, F. and Tamarit, J. (2003). A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computer & Operations Research, 29*(7), 925-947.

Birgin, E. G. Martinez, J. M. and Ronconi, D. P. (2005). Optimizing the Packing of Cylinders into a Rectangular Container: A Nonlinear Approach. *European Journal of Operational Research, 60*(1), 19-33.

Bischoff, E. E. (2003). Dealing with Load bearing Strength Considerations in Container Loading Problems, tech. report, *European business management school, Univ. of Wales, Swansea,* 1-18.

Branch and bound. (2007, October 5). In Wikipedia the Free Encyclopedia. Retrieved 07:53, October 10, 2007, from http://en.wikipedia.org/w/index.php?title=Branch_and_bound&oldid=162399285

Chen, L. Xi, L. Cai, J. Bostel, N. and Dejax, P. (2005). An Integrated Approach for Modeling and Solving the Scheduling Problem of Container Handling Systems. *Journal of Zhejiang University Science A, 7*(2), 234-239.

Cook, J. J. (2004). Language Interoperability and Logic Programming Languages. *Doctor of Philosophy, Laboratory for Foundations of Computer Science, University of Edinburg*, 1-172

Cook, J. J. (2004). Optimizing P#: Translating P# into a more Idiomatic C#. In *Proceedings of CICLOPS 2004*, 59-70.

Faroe, O. Pisinger, D. and Zachariasen, M. (2003). Guided Local Search for the Three-Dimensional Bin Packing Problem. *Informs Journal on Computing, 15*(3), 267-283.

Haag, S. Cummings, M. Phillips, A. (2007). *Management Information Systems for the Information Age*. New York: The McGraw-Hill Companies, Inc.

Neider, J. Davis, T. Woo, M. (1997). *OpenGL Programming Guide: The official guide to learning*. New Work: Addison Wesley.

Knapsack problem (2007, September 1). In *Wikipedia, the Free Encyclopedia.* Retrieved 22:20 September 1, 2007, from http://en.wikipedia.org/wiki/knapsack_problem

Lim, A. and Zhang, X. (2005). The Container Loading Problem. In *Proceedings of the 2005 ACM Symposium on Applied Computing,* 913-917.

Linear programming (2007, October, 1). In *Wikipedia, the Free Encyclopedia.* Retrieved 13:47, October 2, 2007, from

http://en.wikipedia.org/w/index.php?title=Linear_programming&oldid=161551252

Local search (optimization) (2007, July 10). In Wikipedia, the Free Encyclopedia. Retrieved 7:00, October 10, 2007, from http://en.wikipedia.org/w/index.php?title=Local_search_%28optimization%29&oldid=143744314

Martello, S. Pesinger, D. and Vigo, D. (2000) The Three-Dimensional Bin Packing Problem. *Operations research, 48*(2), 256-267.

Metaheuristics. (2007, October 7). In *Wikipedia, the free Encyclopedia.* Retrieved 06:33, October 10,2007, from http://en.wikipedia.org/w/index.php?title=Metaheuristic&oldid=162973403

Pisinger, D. (1999). Core Problems in Knapsack Algorithms. *Operations research, 47*(4), 570-575.

Pisinger, D. (2002). Heuristics for the Container Loading Problem. *European journal of operational research. 141*(2), 382-392.

Pureza, V. and Morabito, R. (2006). Some Experiments with a Simple Tabu Search Algorithm for the Manufacturer's Pallet Loading Problem. *Computers & Operations Research, 33*(3), 804-819.

Scheithauer, G. (1999). LP-based Bounds for the Container and Multi-Container Loading Problem. *International Transactions in Operation Research, 6*(2), 199-213.

Scheithauer, G. (1992). Algorithms for the Container Loading Problem. In *Operations Research Proceedings 1991*, 445-452.

Tabu search. (2007, September 10). In *Wikipedia, The Free Encyclopedia.* Retrieved 11:48, October 6, 2007, from http://en.wikipedia.org/w/index.php?title=Tabu_search&oldid=156994516

Vasquez, M. and Hao, J. (2001). A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem. In *Proceeding of the 17th International Joint Conference on Artificial Intelligence*, 238-233.