The Optimisation of Unitising Designs

Andrew Shewring
Computer Science Department
The University of Auckland
Private Bag 92019, Auckland, New Zealand
a.shewring@gmail.com

Hans W. Guesgen
Institute of Information Sciences and Technology
Massey University
Private Bag 11222, Palmerston North, New Zealand
h.w.guesgen@massey.ac.nz

Abstract

This paper describes the design and development of an application used to determine the most suitable way in which to pack a large quantity of small manufactured product items onto pallets for transportation. A novel constraint-based technique is used in the optimisation to allow for flexible specification of optimisation goals and ranking of solutions.

Introduction and Motivation

Manufacturers need to ensure that their products are delivered to customers in good condition, with the lowest possible cost and human effort. This paper describes an automated system which computes recommended packing configurations to allow employees to focus more on high-level operational matters than the low-level details of constructing suitable units for transportation. In many cases, the creation of packing configurations is still done manually, based on employees' intuition and past experience.

In this paper, a *cargo unit* refers to a unit built from product items supported by a single pallet. A *unitising design* is a collection of cargo units containing the product items required for a customer's order.

Our requirements for this project were as follows:

- Overhang and underhang of product items is to be minimised, as is the wasted space within each cargo unit.
- Wasted space in the assigned container should be minimised.
- The units must be simple for people to pack, and the finished configurations should be displayed to the user.
- The units are to be stable.

Approach

We have adopted a flexible strategy for solving this optimisation problem, based on constraint satisfaction techniques. The relative importance of each factor in the problem, such as underhang or wasted space in the width of the container, is expressed by an associated *cost function*. These functions specify precisely how penalties are assigned to each factor.

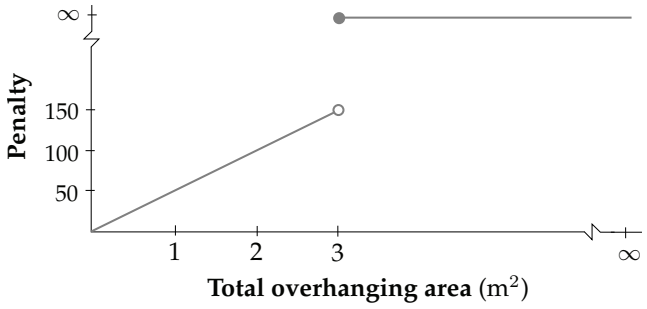

Figure 1: A cost function containing a hard constraint.

Our scheme has much in common with the *constraint hierarchies* [2] described by Borning et al., but the degree of preference for a soft constraint on a factor is implied by the definition of its associated function, rather than being given explicitly. Any constraints deemed invalid or unimportant can be disregarded by setting their functions to a constant value of zero. Figure 1 shows an example cost function with a hard constraint, where the user has chosen to penalise overhang at 50 units per square metre, provided that total overhang is less than three square metres.

For our problem, each *handling unit* (a unit small enough for handling by a person) is composed of a specific number of product items of a fixed size, stacked one on top of the other. The optimisation algorithm must determine the optimal number of product items in the stack. Because the product items are quite flexible and could be easily damaged if packed on their ends, we only allow them to be packed 'flat', or face down. Handling units are assembled into configurations called *patterns*. Some manufacturers adopt a set of standard configurations. The graphical editor depicted in the background of Figure 2 is used to manage the available patterns. Those patterns which are deemed unstable or less suitable than others may be assigned penalties by the user. The system accounts for the likely wasted space in the rest of the container in which the customer order is to be packed, without knowledge of other orders, by solving an instance of the Unbounded Knapsack Problem. Our method is an extension of an algorithm described by Brassard and Bratley [3] (Chapter 9, page 308).

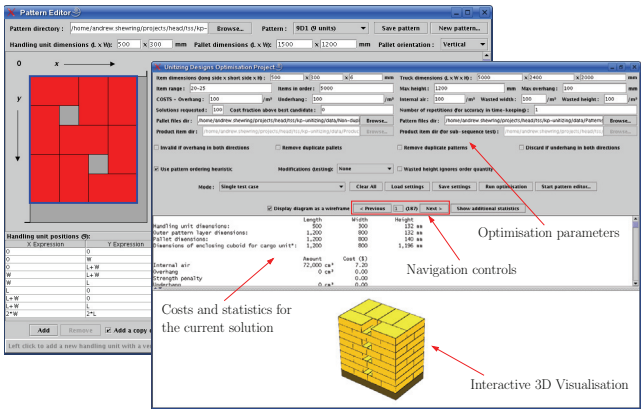

Figure 2: A screenshot of the user interface components.

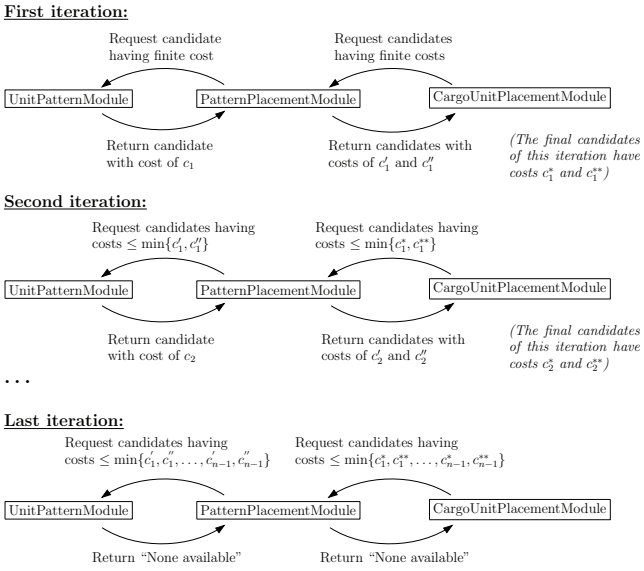

Figure 3: Communication of modules during optimisation.

Branch and bound [1] is a technique used in combinatorial optimisation to prune the search tree. There is a clear hierarchy between each of the components created by the packing process: cargo units are composed of packing patterns, which are composed of handling units, which are in turn composed of product items. This hierarchy provides the tree of sub-problems (branching). Designs are discarded from consideration in the bounding step if their penalties exceed those of other designs at later stages in the process.

Main Components and Algorithms

Three main modules in the system are used to create the different types of *candidates* (partial solutions). These are *UnitPatternCandidate*s, *PatternPlacementCandidate*s and *CargoUnitPlacementCandidate*s, and correspond to concrete packing pattern layers, individual cargo units and multiple cargo units (completed unitising designs), respectively.

The modules communicate via a request-response protocol, as shown in Figure 3, and collaborate to build candidates with minimal total penalties. After each iteration, the lower bound on penalties is adjusted. The end result is a list of units satisfying all constraints and other directives given by the user.

Our system is very suitable for interactive use, as solutions to problems with several hundred available patterns are produced in seconds. The resulting solutions are generally of high overall quality, even when a relatively limited number of patterns and pallets are available.

User Interface

The optimisation problem is set up and solved using the interface in the foreground of Figure 2. It displays an interactive three-dimensional representation of each feasible solution (ranked by total penalty) and associated statistics.

Conclusion and Future Work

This paper describes an application created to optimise the packing of products onto pallets, to construct cargo units. The relative importance of each factor in the optimisation is specified by way of cost functions which control the assignment of penalties. The combination of the layer-by-layer packing strategy and the three-dimensional visualisation of the cargo units in a solution ensure that the cargo units are simple to pack and to visualise. In summary, the system is powerful, easy to use and highly customisable.

Future enhancements could include the use of multiple cargo unit types in each order, potentially improving space utilisation; using algorithmic placement of cargo units (like in Dean et al.'s algorithm [4]) to better estimate wastage; and employing a more sophisticated way of penalising unstable cargo units.

References

[1] P. E. Black. "Branch and bound", from Dictionary of Algorithms and Data Structures. Paul E. Black, ed., NIST. Available from http://www.nist.gov/dads/HTML/branchNbound.html.

[2] A. Borning, R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf. Constraint hierarchies. In Norman Meyrowitz, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 22, pages 48–60, New York, NY, 1987. ACM Press.

[3] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, Inc., NJ, USA, 1996.

[4] H. T. Dean, J. N. Baggaley, and R. J. W. James. Three Dimensional Container Packing of Drums and Pallets. In *ORSNZ '99: Proceedings of the 34th Annual Conference of the Operational Research Society of New Zealand*, Hamilton, New Zealand, 1999.