Analysis of Bin-packing Algorithms Used for Steel Beam Cut Optimization

Michael Brohm
Analysis of Bin-packing Algorithms Used for Steel Beam Cut Optimization

Michael Brohm
Dept. of Computer Science
University of Trollhättan/Uddevalla
P.O. Box 957, SE-461 29
Trollhättan, Sweden
michael@bhm.se

Abstract

The increasing steel price makes a tool with a high quality algorithm to reduce steel beam waste more important.

The cut-optimization of stored steel beams is a classical cut-optimization problem with one exception, because beams are stored in several different lengths, the algorithm should not only deal with where a beam should be cut it should choose the appropriate length of stock beam to use as well.

StålSpec 2000 is an application used by engineering workshops and companies which develop, manufacture or sell metal goods or products. The feature of the application discussed in this paper is the optimization of steel beam cuttings or more specifically the optimization algorithm. StålSpec 2000 is migrating from legacy procedural Visual Basic 6 code to modern component based C# code and it is not obvious which algorithm to use the new rewritten version of StålSpec is used as a case study.

The effectiveness of five bin-packing algorithms was analyzed in the case study to find the most appropriate one to use.

The case study result pointed out the current algorithm in StålSpec 2000 to be the best choice. The next rewritten version of StålSpec is going to contain a similar algorithm as its predecessor.

1. Introduction

The Swedish engineering workshops and companies which develop manufacture or sell metal goods or products have their own trade organisation called MVR (Mekaniska Verkstädersnas Riksförbund). MVR is marketing software called StålSpec 2000 to its members for creation of specifications used for ordering material, calculating weights, prices, transportation costs etc. One module in the application is steel beam cut optimization. The goal is to reduce waste in form of leftovers from steel beams or pipes. [1]

For example if a company is about to build a steel structure for a warehouse. The construction engineer simply takes the warehouse blue prints and enters all beams types, dimensions and lengths into the application. The application then uses the cutting optimization algorithm to calculate the best way to cut the beams to make a more cost effective order from the steel supplier.

The cutting optimization problem in StålSpec 2000 differs from the original definition of a cutting stock problem. Steel suppliers’ stores steel beams in different lengths so the algorithm in StålSpec has to deal with multiple sizes of bins and it is up to the algorithm to choose the most efficient lengths to use.

Steel prices are rapidly increasing as shown in figure 1.1, mainly because of the increasing demand for scrap steel (which beams are made out of ) on the Chinese and Indian market but also because more steel is stored in stead of manufactured on demand [2, 3]. The reason for storing material instead of manufacturing it is that the large number of small steelworks has been closed or merged with larger companies and to manufacture small batches of steel is simply not profitable. To store a large variety of steel products is costly because of the long time before the invested capital is returned.

When prices increase the importance of a tool to reduce cost also increases.
StålSpec 2000 is migrating from legacy procedural Visual Basic 6 code to modern component based C# .NET code and since the bin-packing algorithm is an optimization algorithm the result of it varies in quality it is important to know what algorithm to use.

The algorithms discussed in this paper are called bin-packing algorithms that deal with a cutting stock problem. The goal is to pack a collection of objects into the minimum number of fixed-size "bins" or to minimize the number of objects to fill an amount of bins, it is a classic hard problem for which many different solutions have been proposed [6]. The bin-packing algorithm is a so called optimization algorithm which is different from the previously mentioned sorting algorithms when it comes to the result, in that the result could differ in quality depending on what algorithm that is used.

There are several different types of bin-packing algorithms and they could work with one two or three dimensional objects. [6]

Deterministic Polynomial-time problems are problems where the algorithm has a running time that could be described by a formula, ex: \( T = 3n \) in this case the time \( T \) is linearly increasing with the number of items \( n \). When it is impossible to determine the running time the algorithm is said to be Non-deterministic Polynomial-time (NP). [7, 8]

The simplest bin-packing algorithms are called online algorithms; they sequentially process a list of items and when an item is placed in a bin it is never moved. A more effective way of optimizing the result could be to test a large amount of different suggestions on how the items should be placed. In this paper, we have adopted the terminology used in [9], and call each different suggestion a 'pattern'. [10]

Some algorithms that deal with the optimization problems are of the type called 'Genetic Algorithms'. "Genetic algorithms are inspired by Darwin's theory of evolution. Solution to a problem solved by genetic algorithms uses an evolutionary process (it is evolved)."[11]

There are several areas of computer science and technology where the bin-packing algorithm is a necessity; for instance in computers memory allocation for efficient data storage, in wireless data communication to fit as many indivisible data blocks in as few timeslots as possible or cutting optimization to reduce waste [12].

3. Method

To answer the question if there is a single best algorithm to use for steel beam cut optimization five different cut-optimization algorithms was implemented in a test suite and actual data from one of StålSpec 2000 users was entered. [13] The test involves measuring the waste of each cut-optimization using all algorithms and analysing the result of each of the cases where the result differed.
4. Algorithms – Theoretical background

The way that the algorithm acquires data tells us if it is an online or offline algorithm. If the algorithms is able to process items and generate an answer before all items are known the algorithm is said to be an online algorithm. And consequently the algorithm that has to know all items before starting to process them is an offline algorithm.

4.1 Online Algorithms

Online Bin-packing algorithms place items in the bins in sequence; one item has to be placed before the next is processed. A drawback of the online algorithms is that no matter if the algorithm is allowed unlimited computation, the result will not change because when an item is placed it can not change its decision. Another major drawback with online algorithms is the problem with packing large items, especially when they occur late in the input. One way around this problem is to sort the items in a decreasing order. [10]

4.1.1. The Next Fit algorithm is the simplest of the ones discussed in this paper. The algorithm sequentially processes the items and if the item fits in the bin containing the previous item it is placed there, if not a new bin is created and the item is placed there. This algorithm is simple to implement and it runs in linear time. The performance of the algorithm is very good but the quality of the bin packing is not. The worst case scenario shown in figure 4.1 is however not too bad, since bini + bini+1 are always going to contain more than the total capacity of one bin. [10]

4.1.2. The First Fit algorithm sequentially processes the items and places them in the leftmost bin that still has room. If no bin can hold the item a new bin is created and the item is placed there. The effectiveness of the First Fit algorithm varies greatly depending on whether the items are sorted or not. If the items are sorted the algorithm is called First Fit decreasing or First Fit nonincreasing. The reason for calling it nonincreasing instead of decreasing is that if there is more than one item of the same length it will not be a decreasing order. I will refer to the algorithm First Fit decreasing in this paper. First Fit guarantees a maximum of one bin to be more than half empty. [5, 10]

In figure 4.2 we can easily see the disadvantage of online algorithms; the packing is not fully optimized, and the items can be packed into three bins instead of four.

4.2 Offline Algorithms

If the algorithm is allowed to view the entire item list before the presenting an answer we should expect a better result. By exhaustive search the optimal packing could be found. [10]

4.2.1. The Gilmore-Gomory cutting-stock heuristic is an interesting approach to the problem. Without the drawbacks of the online algorithms several different patterns could be tested and the optimal one applied.

The Gilmore-Gomory approach uses the following data: \((L, s, n, i, m, B)\), \(L\) is a list of size / quantity pairs.
(s_i, n_i), 1 ≤ i ≤ m. An unsigned integer vector \( p = (p_1, \ldots, p_m) \) is used to represent a packing pattern if all packed items is within the bin size (\( B \)). [9, 14, 15]

The Gilmore-Gomory cutting-stock method uses column generation to reduce the amount of possible patterns. The algorithm starts with a small initial set of items which grows dynamically in size. Items are added and different combinations of patterns are tested to find the most suitable combination. [9]

![Figure 4.3 Result of applying Gilmore-Gomory. Item sizes: 20, 85, 50, 20, 40, 30, 40, 10. Bin size: 100. Waste: 1.7%](image)

4.2.2 The genetic algorithm approach is a way of optimizing the result through evolution. The algorithm works in several steps and it often starts with an initiation phase where a set of different patterns or initial solutions are presented, this could be generated by simpler online bin-packing algorithms like First Fit or Best Fit or it could randomly created.

The result of the initiation should be a collection of patterns called an initial population and each pattern could be seen as a chromosome. The initial population contains too many chromosomes to undergo the iterative steps of the algorithm, so a natural selection has to be made and only the best is selected to a mating pool. The next step is to pair the chromosomes. There are a variety of different methods and an easy one is where you pair the best chromosome with the second best and the third with the fourth and so forth. When the pairing is done a creation of one or more offspring is made, this is called the mating phase. In the next step random changes or mutations are made to a small percentage of the chromosomes. The mutations are the second way the algorithm changes the pattern to optimize the result. The old population and the offspring build a new population and where back at natural selection to the mating pool. [11, 16]

![Figure 4.4 Iterative process of a genetic algorithm](image)

4.2.3 The algorithm used in the legacy code of StålSpec 2000 contains an offline algorithm for cut-optimization. What makes the StålSpec algorithm unique is its ability to handle multiple bin sizes. The packing algorithm itself is not an offline algorithm; the online algorithm Best Fit is used. The part of the algorithm that makes it an offline algorithm is the exhaustive testing of different bin size patterns. Best Fit Decreasing is run on every possible bin size pattern. After the bin packing is done the bins are trimmed if it is possible. The efficiency of StålSpec bin packing algorithm is never worse than Best Fit Decreasing since it is the algorithm used. [17]

The result of applying the StålSpec algorithm on item sizes 20, 85, 50, 20, 40, 30, 40, 10 and bin size 100 is the same as for First Fit decreasing in figure 4.2.

5. The legacy code - Software engineering considerations

StålSpec 2000 is migrating from the structured programming language (Visual Basic 5) to the object oriented programming language C# .NET. The reason for this change is that the application contains a variety of bugs and to bug fix the 41000 lines of source code would be a waste of time because the whole architecture of the application is poorly built, and no legacy design documentation exists. The problems with StålSpec 2000 are classical problems when a small program gradually grows larger. Other classic problems that occur in StålSpec 2000 and often arise when there is only one single developer in a project, is
lack of documentation, code comments (less than 1%) and lack of a proper naming standard for functions, variables and classes. The extensive use of global variables makes the source code in some cases hard to comprehend. The versions of StålSpec 2000 out on the market has been randomly bug fixed on the spot when customers has encountered problems and asked for support, this way numerous versions of the application has been created.[1] The variety of versions is also a reason to make a new application instead of bug fixing the old one. It is hard to support several different versions of a system when a versioning system like CVS has not been used.

The question is what kind of algorithm is best suited for the cut optimization component in the new rewritten StålSpec. The reason for analysing several different algorithms when there already is one who deals with the problem is that the migration from a procedural Visual Basic 6 code to the object oriented language C# .NET is not a trivial task. An adoption of the legacy algorithm was shown to be harder than we expected, the over use of global variables made a natural adoption impossible due to high dependencies. Extensive modifications of the code could make it useable in the new C# version of StålSpec, but because immense changes has to be made a completely rewrite is a better choice. Since the algorithm is going to be rewritten it could be wise to look at different algorithms.

6. Qualitative discussion of the candidate algorithms

One aspect we must consider is the quality that has to be delivered by the algorithm. After a quick glance at the online algorithms we could conclude that their result is not good enough. The worst case result of the Next Fit algorithm is even worse than letting the workers pick their beams at random. This enables us to rule out the Next Fit algorithm as a single best algorithm to use, the algorithms could however be used to initialize offline algorithms. The First Fit decreasing algorithm and the Best Fit Decreasing algorithm is much better than Next Fit but it is still not optimal. The example in figure 4.2 could be made much better as seen in figure 4.3.

The major benefits of online algorithms are the performance and the fact that the bin-packing could start before all items are known. [10] Due to the fact that performance is not a major issue in this case and that all the items always going to be known the benefits of the online algorithms are not worth much.

As mentioned before the bin-packing algorithm of StålSpec is able to make decisions about what bin sizes to use and that capability is not available in any online algorithm at the moment. Even if an online algorithm was modified to handle multiple bin sizes it would be impossible to make it efficient due to the fact that all items must be known to make such a decision.

Offline algorithms have much better conditions for making an optimal result. Instead of focusing on placing items in the right bin at the first time we could let the CPU take care of the problem by trying multiple different suggestions. The offline algorithms are NP and need more time to make the optimization, this is however not a problem in this case since the amount of beams in a cut optimization is limited. The number of beams to cut in an average StålSpec 2000 cut optimization is less than ~100.[13] One must however bear in mind that computers in workshops not always has the performance and capacity of retail computers.

As previously mentioned steel suppliers store the beams in different lengths. This means that the algorithm should not only decide where to make the cuts on a stock beam it should also choose the most appropriate length from the stock. The legacy algorithm is the only algorithm that handles this problem. One up to five different lengths could be used in a single optimization.

The Gilmore-Gomory cutting-stock method is of course a better algorithm to use when there is one fixed size bin. The simple example in figure 41-3 clearly shows that Gilmore-Gomory method has an advantage that online methods lack.

If the Gilmore-Gomory method should be adapted to StålSpec it has to handle multiple bins sizes, and that’s a problem. The packing algorithm inside StålSpec is Best Fit Decreasing and to test different bin patterns with Best Fit is different than testing different bin patterns to any other algorithm mentioned in the paper. If a beam is available in 3 different lengths at the supplier, Best Fit only needs to be run 7 times since the algorithm always scans all bins before it is placed in one.

Table 6.1 Possible bin patterns for Best Fit when three different lengths are available.

<table>
<thead>
<tr>
<th>Length A</th>
<th>Length B</th>
<th>Length C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>
The Gilmore-Gomory method does not only need to be tested with all possible combinations of bin lengths, it needs to be tested in different order to. Since the quantity of unique beam patterns is increasing exponentially with the amount of items to pack, and the quantity of unique bin patterns is increasing exponentially with the amount of bins to use, the total amount of unique patterns is going to increase even more. Performance suddenly becomes a problem. [9, 14, 15]

The genetic algorithm has the same advantages and disadvantages as the Gilmore-Gomory method, because it works in much the same way. The difference is that the Gilmore-Gomory method only mutates the chromosomes. The performance problem is going to arise if the algorithm is modified to handle multiple bins. [18, 19]

7. Requirements

To insure us that the new version of StålSpec does what it was suppose to we gathered information at five different meetings that where held by MVR or by MVR members on locations where StålSpec is used.

At the first meeting with the MD of MVR we acquired the source code of StålSpec 2000. At first the task was to correct bugs in the source code that has haunted user of StålSpec for years, but it had been proved to be a bad solution since a company who tried it failed to remove enough bugs to make it stable. The MD of MVR wanted us to rewrite the whole application in a modern language. The choice of programming language fell on C-Sharp C# because it is a modern language that is type safe which means that no unsafe casts can be made, it is object oriented which means data encapsulation, inheritance, polymorphism, interfaces and it is scalable and updateable through its support of component based design. [20]

The source code of StålSpec 2000 was meant to serve as a requirement specification but as mentioned before the code of was hard to comprehend and the extensive use of global variables made it hard to understand it. A fact that made it hard to rewrite StålSpec was that not all was keen on us doing it. We contacted the author of StålSpec to ask him if he used any known bin-packing algorithm when he implemented the cut-optimization function. The response was a single line e-mail where he stated that he used a completely own construction of how to optimize the steel beams. After studying the content of variables in Visual Studio version 6 debug mode it showed that different bin sizes where tested repeatedly in other words an offline algorithm. When we tested the algorithm with different beam collections with a single bin size it proved to have the same result as Best Fit Decreasing.

The second meeting where held at MVR in Stockholm with a reference group. The reference group is made up by the three users of StålSpec and the MD of MVR. The reference group informed about the parts of StålSpec that could be excluded in the new version and parts that needed extra attention. The cut-optimization was one of those. The algorithm should obviously not perform worse than the old algorithm when it comes to waste reduction. A desire from Bröderna Edstrands AB was that the new version should handle more than five different stock beam lengths since several beam types where stored in more than five lengths. [2]

The third and fourth meetings were held in Upplands Väsby and in Norrköping. The way StålSpec 2000 is used was demonstrated. We got acquainted with some of the bugs in StålSpec 2000 when the CEO of Bröderna Edstrands AB was demonstrating the cut optimization, the beams to cut was in was shown with over 90% waste. The CEO of MVR emphasized the need for bug free software.

The last meeting where held at the MVR congress in Tellberg. We had the opportunity to meet several users of StålSpec and gather their point of views. The most common thought was that the StålSpec 2000 was a great application that is much needed but that the bugs made them uncomfortable and forced them ignore some parts of the application to prevent it from crashing. Some people at the congress had difficulties concealing their thoughts about StålSpec as it has made them furious a couple of times.

8. Implementation

Microsoft Visual Studio .NET 2003 was used to implement First Fit Decreasing, Next Fit Decreasing, Best Fit Decreasing and the Gilmore-Gomory algorithm in a single application to measure their waste rate on actual cut optimizations. The data that was used came from one of the companies in the reference group.

The original StålSpec 2000 was used to test the legacy algorithm. A specification was created with the beam lengths and then cut-optimized. The legacy algorithm and the genetic algorithm was initially meant to be implemented in the test suite but had to be left behind due to lack of time.
8.1 Online algorithms

The online algorithms are almost identical designed; the difference between First Fit Decreasing and Next Fit Decreasing is that the First Fit algorithm goes back to the first bin after placing an item in a bin, in other words this row.

```java
binIndex = 0;
```

The data input is made by reading a text file with cut-optimization info, bin sizes and item sizes. Arrays representing the bins and beams are created. One additional array is created to hold the available bin size. The beams are sorted in descending order. The algorithm loops through all beams and puts them in the current bin. If it is the Best Fit algorithm it puts it in the tightest bin, if it is the Next Fit algorithm it puts it in the previous bin if it has room else it is placed in the next bin, and the First Fit algorithm places it in the first bin that has room.

8.2 Gilmore-Gomory algorithm

The Gilmore-Gomory algorithm uses the same variables as the online algorithms. Two additional arrays are used, one to hold different patterns and one to hold the score of each pattern. The pattern is in fact the order of beams to cut.

The algorithm starts by using First Fit Decreasing to generate an initial pattern and thereby preventing it to be worse than First Fit. The pattern array is then filled with 250,000 randomly changed patterns. Next Fit is used to calculate a score and the best pattern is returned and applied.

The column generation algorithm is however not implemented due to lack of time. The benefit of column generation is not crucial in the test cases since they are so small.

8.3 The Genetic Algorithm

The genetic algorithm was never implemented due to lack of time, but it would not make much difference because the algorithm has the same disadvantage as the Gilmore-Gomory algorithm.

The algorithm would start by creating a number of random patterns about 10,000. The top 200 of these will be naturally selected to a mating pool, no duplicates. The patterns or chromosomes as they are called would be paired two and two. The chromosomes would be compared and parts where they are alike would be “locked” or ignored and a random part of the chromosome where they are different would inherit to their offspring. This would be repeated until the chromosomes or population is 5,000.

**Figure 8.1 Chromosome B inherit position 5 from Chromosome A, giving Offspring A**

![Figure 8.1 Chromosome B inherit position 5 from Chromosome A, giving Offspring A](image)

After the mating phase the offspring would be randomly altered, exactly as they are in the Gilmore-Gomory algorithm. When the population reaches 10,000 the natural selection is made again. The procedure would end if it reached 0% waste, the loop limit was exceeded or when the result didn’t get any better. [16]

8.4 Development method

The new rewritten version of StålSpec is developed by using a hybrid of two well known development techniques namely Unified Process (UP) and Extreme Programming (XP). [21] The hybrid development method was used because we where three developers working on a rather unique software and not all aspects of the two established development methods suited your particular situation.

To avoid ending up with versioning problems like StålSpec 2000 we use a CVS repository. The CVS repository is also used because we work on the same code base.

The test suite however is a very simplified version of StålSpec without beam drawing or any fancy graphics and was developed with a type of prototyping method [22, 23]. The prototypes were built from algorithm flowcharts and gradually improved until they functioned properly.

9. Test case results

To be able to evaluate the algorithms suitability as an algorithm for steel beam cutting optimization, actual data from a company using StålSpec on a daily basis was used to test the algorithms and the result is shown in figure 9.1. The thirteen test cases were picked at random from the StålSpec user [13], optimization with less than three items where left out. The items to cut varied in quantity, the largest optimization contained...
196 items but where fairly easy to optimize since the items where small (180 – 350mm), the smallest optimization contained only four beams. The average quantity of items to optimize in the cases was 34. Average waste for all cases is shown in table 9.1.

Test cases with different waste result depending on what algorithm was used is viewed a bit closer.

9.1. Case number one has fifteen beams to optimize, the beam lengths are from 5.5m to 11m. The available lengths from the supplier are 10, 12, 14 and 15m.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Fit Decreasing</td>
<td>10%</td>
</tr>
<tr>
<td>Next Fit Decreasing</td>
<td>10%</td>
</tr>
<tr>
<td>Best Fit Decreasing</td>
<td>10%</td>
</tr>
<tr>
<td>StålSpec 2000</td>
<td>7%</td>
</tr>
<tr>
<td>Gilmore-Gomory</td>
<td>10%</td>
</tr>
</tbody>
</table>

Table 9.2 Beams to cut in test case number two

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEA 140</td>
<td>2500</td>
<td>1</td>
</tr>
<tr>
<td>HEA 140</td>
<td>3000</td>
<td>12</td>
</tr>
<tr>
<td>HEA 140</td>
<td>3500</td>
<td>1</td>
</tr>
</tbody>
</table>

The optimization is again simple, 10 of the 3m beams are cut from two 15m stock beams leaving no waste, the rest is cut from one 15m stock beam leaving 3m waste except for the StålSpec algorithm that cuts the remaining beams from a 12m stock beam with no waste. Once again the waste could be avoided by trimming the stock beams.
**Test case number four** is particularly interesting because of the huge difference in the result. Like the other optimizations the available lengths from the supplier are 10, 12, 14 and 15m. The beams to cut have a larger length difference than the earlier optimizations, from 3 to 12,5m.

<table>
<thead>
<tr>
<th>Table 9.3 Beams to optimize in test case number four.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

**Figure 9.2 Items in test case number four.**
A. Beams to cut optimize.
B. Typical result using fixed size bins.
C. Result using multiple sizes of bins.

Test case number four is a good example when multiple bin sizes are most effective to use. As shown in table 5.1 the longest beam are 12,5m, which means that the algorithm using one fixed size bins has to use a larger one for the whole optimization, in this case 14 or 15m. The use of a size 14m bin gives a total waste of 20% which is a bad result compared to 3% for the StålSpec algorithm. This time a simple trim of the stock beams after the algorithm was done would improve the result but not to the extent as one might hope. If the trim function where a part of the general algorithm it could generate the same result as StålSpec 2000, but that only works for the offline algorithms Gilmore-Gomory and the Genetic algorithm if it was implemented.

**9.1.3 Case number ten** is a simple optimization, but it is interesting from another point of view. One might be deceived, there are 22 beams to optimize and they are all 2495mm in length. The available stock beam lengths from the supplier are 10, 12, 14 and 15m. If no optimization tool is used one could make the mistake off thinking that four 2495mm beams on each 10m stock beam are the best choice. But that would leave only two 2495mm beams on the last 10m stock beam and give total waste of 5110mm or 9% as all algorithms except StålSpec 2000 did. If three 15m and one 10m stock beam where used instead the waste would be 110mm or 0,2%.

**10. Conclusion**

The study shows that research has been done on bin-packing with multiple bin sizes but the articles found on IEEE Search and the Internet do not fully apply to the need that the new version of StålSpec has. [18, 24] The exhaustive algorithms often have a running time up to 15 minutes and that is not acceptable. [18]

The study has shown that there are things that could have been done differently with a slightly better outcome. A modification of the well-known algorithms would make the result equal to the legacy algorithm in some parts. A simple trim function would improve the result in all but one test case.

The study shows that the ability to handle multiple bin sizes is clearly the greatest advantage when it comes to steel beam cut optimization. The result shows that the Gilmore-Gomory method is not even better than the online algorithms. A build specification might contain a large amount of beams but one has to bear in mind that there is a lot of different dimensions, shapes and steel qualities used in a project which makes each cut optimization relatively small, and the need for advanced decision making in not crucial.

The study also shows that it’s dangerous for programmers to use their own algorithm to solve a difficult problem, unless they have extensively tested and, more importantly, documented it. This is also true in applications like data compression and encryption - many programmers have come up with their own algorithms that have later been proved to be much worse than existing well-tested methods that are better understood at a theoretical level. On the other hand, the number of possible variations on bin-packing algorithms is immense, and the efficiency of algorithms when used with their own 'real-world' data is often more important to the customer than an impressive 'average-case' performance of a well-known algorithm tested with totally random data. Extensive 'on-site' testing and good documentation seem to be the best recommendations. Offline algorithms would also seem to have the benefit that, because they are fast, a number of different algorithms could be tried 'in parallel' on the same data, and the 'winner' used to decide on the cutting pattern.
Based on this thesis the next rewritten version of StålSpec is going to contain a similar algorithm as its predecessor had. I use the word similar because nothing of the old code is going to be reused; it is simply not good enough.

There is always danger that implementing changes in software in an industrial setting will be opposed by some people in the company - it is the company’s responsibility to set clear policies to resolve such problems.

An unfortunate thing is that the members of the reference group and the users meet in Tellberg did not have all the facts. The only person with the information concerning programming, calculation and know-how within our field was not available. Discussing algorithms with people from the steel industry is very difficult, firstly because they are not used to, they calculate and use mathematics regularly but it is still not algorithms, and secondly because they don’t care as long as it works. If an experienced steel worker would join the project group and participate closer to the development some of the issues have a better chance to be solved.

11. Suggestions for Further Work

To give suggestions of further work on a subject that has been struggled with for over fifty years is not easy. One area within bin-packing that not seems to be researched a lot is the problem with multiple bin sizes. [24] It is not strange because in most definitions of the problem specifies that the bins are of a fixed size, mainly because these simplified cases are easier to treat theoretically.

12. References


[17] B-G. Lundgren, Stålbyggnadsteknik AB, Personal Correspondence 2004-04-26


