Hybrid Heuristics for Planning Lot Setups and Sizes

Alistair R. Clark

Faculty of Computing, Engineering and Mathematical Sciences
University of the West of England, Bristol, BS16 1QY, England
Tel: +44 (0)117 344 3134. Fax: +44 (0)117 344 2734
Email: Alistair.Clark@uwe.ac.uk

Accepted for publication in
Computers and Industrial Engineering

June 20, 2003

Abstract

The planning of a canning line at a drinks manufacturer is discussed and formulated as a mathematical programming model. Several alternative heuristic solution methods are developed, tested and compared on real data, illustrating the trade-offs between solution quality and computing time. The two most successful methods make hybrid use of local search and integer programming, but in rather different ways. The first method searches for the best proportion by which to factor setup times into unit production times. The second method carries out a local search on the first stage’s binary setup variables. In both methods approximate mixed integer programming models are solved at each search iteration. In addition, a local search variant, called diminishing neighbourhood search, is used in order to avoid local optima in a variety of landscapes. Computational tests analyse the quality/time trade-offs between alternative heuristics, enabling an efficient frontier of non-dominated solutions to be identified.

Key Words: Production planning; Lot sizing; Setups; Heuristics; Optimisation; Local search.

© 2003 Alistair Clark, All Rights Reserved
1 Introduction

This paper explores several different heuristic solution approaches for a prototype mixed integer linear programme (MIP) model developed to assist a drinks manufacturer plan the canning of a range of liquid products. The company also bottled these products, but selected the canning line as appropriate for a pilot project, since the line was dedicated to canned products which, in turn, were not produced on any other line. Thus the planning and scheduling of the canning line and its products was carried out independently of the company’s bottling lines and associated products.

A concern of the company was to assess the impact of sales promotion commitments and rushed orders on both finished inventory and its ability to fulfil customer orders. The intention was that the model would enable the company to explore the impact of possible promotions and customer order scenarios while using capacity as efficiently as possible. The model sought to optimize the fulfillment of customer orders, taking into account forecast demand, the capacity of the canning line, and the limited availability of liquid mixing tanks, as well as a technological constraint that the tanks should never be used less than half-full. The planning (and scheduling) also had to take into account the impact of product changeovers (setups) on the effective capacity of the canning line.

As the forecasts of individual customer orders and product-specific demand were quite error-prone, there was little point carrying out detailed scheduling for more than a week in advance. However, it was useful to have a provisional production plan for thirteen weeks ahead in order to have advance warning of possible backlogs. Accordingly the proposed system had the following outputs:

1. A production schedule for week 1, detailing which product should be produced on the canning line, their batch sizes and in what sequence.

2. A provisional plan showing the amount produced of each product in each of weeks 2 to 13 in the first three months. Many products will not be produced every week if scarce capacity is to be used efficiently.

If a large number of products had to be sequenced each week, then a mathematical algorithm would almost certainly identify a more efficient production sequence than a human scheduler. However, the straightforward nature of changeover times meant that the sequencing of products was well handled by the experienced scheduler who simply sequences together products that used the same liquid. Consequently this paper focuses on the planning model.

Since much demand data is forecast and some production parameters are not precisely known, a solution that is optimal for estimated data may actually add less value than a near-optimal solution. Indeed, it will almost certainly be sub-optimal for the actual data. Rather than seek a perfectly optimal solution, the aim was to find a good solution in practicable time. Thus the objective of this paper is to explore and compare the quality and computing time of several alternative solution approaches. Its purpose is to inform both the academic and practitioner industrial engineering community of the interesting and useful results obtained in this particular case.

The structure of the paper is as follows. Section 2 formulates and explains the planning model and then in Section 3 algorithmic solution methods using only mathematical programming are developed and computationally tested. In Section 4 hybrid heuristics combining local search and mathematical programming are explored and also tested. Finally, Section 5 concludes the paper with a comparative discussion of the test results and provides
some pointers for further research. The main conclusion is that, for the situation and data of the drinks manufacturer, hybrid approaches comprising both classical mathematical programming and modern local search methods were the most successful in balancing solution quality with computing time.

2 The planning model

The plant canned $P = 41$ distinct products $p$ filled from $L = 14$ liquids $l$. The production of both liquids and canned products was planned on a rolling horizon basis for each of $T = 13$ consecutive weeks $t$. A basic formulation of the planning problem is:

$$\text{Minimise } \sum_{p,t} \left( I^+_p + 100I^-_p \right)$$

such that

$$I^+_{p,t-1} - I^-_{p,t-1} + x_{pt} - I^+_p + I^-_p = d_{pt} \quad \forall \ p, t \quad (2)$$

$$\sum_p a x_{pt} + s \left( \sum_p y_{pt} - 1 \right) + e \left( \sum_l z_{lt} - 1 \right) \leq B_t \quad \forall \ t \quad (3)$$

$$y_{pt} \leq z_{lt} \quad \forall \ l, p \in P(t), t \quad (4)$$

$$x_{pt} \leq M_{pt} y_{pt} \quad \forall \ p, t \quad (5)$$

The decision variables are:

- $x_{pt}$ Quantity of product $p$ produced in week $t$, ($\geq 0$)
- $I^+_p$ Stock of product $p$ at the end of week $t$, ($\geq 0$)
- $I^-_p$ Backlog of product $p$ at the end of week $t$, ($\geq 0$)
- $y_{pt}$ Binary variable taking value 1 if canned product $p$ is produced in week $t$, and 0 otherwise.
- $z_{lt}$ Binary variable taking value 1 if liquid $l$ is produced in week $t$, and 0 otherwise.

The parameter and data inputs are:

- $d_{pt}$ Demand for product $p$ at the end of week $t$.
- $B_t$ Available time on the line in week $t$.

- $a$ Canning-line time required to produce one unit of any product, excluding changeovers.
- $s$ Canning-line setup time needed to changeover between canned products if no change of liquid is involved.
Extra canning-line setup time needed in the changeover between canned products if a
change of liquid is involved.

\[ P(l) \]  Set of canned products that use liquid \( l \).

\( M_{pt} \)  Upper bound on \( x_{pt} \), calculated as \( b_t/a_p \).

The objective function (1) minimises inventories and backorder penalties, the latter hav-
ing much more weight than the former, so as to flexibly but strongly discourage planned
backorders rather than outrightly prohibiting them. The first constraints (2) balance pro-
duction, inventories and backlogs with demand. The stock \( I_{pt}^+ \) and backlog \( I_{pt}^- \) cannot both be positive for a given pair \( (p,t) \), an occurrence that is prevented by both having positive coefficients in the objective function (1). Many products have positive (negative) current stocks which, in a data preparation phase, are discounted from (added to) demand in the first period(s), resulting in zero current stocks and what is known as effective demand, calculated as follows:

\[
\text{For } t = 1 \text{ to } T \text{ do }
\begin{align*}
\text{Let } d_{pt} &= \max (d_{pt} - I_{p0}, 0); \\
\text{Let } I_{p0} &= \max (I_{p0} - d_{pt}, 0); \\
\text{If } I_{p0} = 0 \text{ then stop;}
\end{align*}
\]

\[ \text{(end of } t \text{ loop)} \]

If \( I_{p0} > 0 \) then do not plan for product \( p \).

where \( I_{p0} \) is the current stock (positive or negative) of product \( p \).

Constraints (3) ensure that production and setups take place within the available time.
The -1 terms in (3) reflect the fact that the first product of the week is either a continuation
from the previous week or a new setup carried out on the Sunday in which it does not
consume canning line time. Constraints (4) and (5) ensure that if a product is canned then
both it and its liquid need to be set up.

The model has been simplified for the purposes of this paper, taking out the availability
of overtime and constraints on the time canned products may stay in inventory before
shipping. However, a key feature was the inclusion of liquid production, carried out in
tanks of a large fixed size \( F \):

\[
\sum_{p \in P(l)} r_{lp} x_{pt} = F n_{lt} \quad \forall \ l, \ t
\]

where \( r_{lp} \) is the amount of liquid \( l \) required to make one unit of canned product \( p \) and \( n_{lt} \) is
an integer decision variable for the number of tanks of liquid \( l \) to produce and have ready
in week \( t \) for canning.

For certain slow-moving liquids, it is not always economic to produce a complete tankful,
and so a continuous variable \( u_{lt} \geq 0 \) is included to quantify the underutilisation of a single
tank of liquid \( l \) in week \( t \):

\[
r_{lp} \sum_{p \in P(l)} x_{pt} = F (n_{lt} - u_{lt}) \quad \forall \ l, \ t
\]

The constraints

\[
u_{lt} \leq 0.5 \quad \forall \ l, \ t
\]
are also necessary since it is not technically viable to produce less than about half a tankful of liquid.

The use of the tanks also needs scheduling each week, but to avoid an overly complex model a capacity limit of \( C_t \) tankfuls in week \( t \) is simply imposed:

\[
\sum_l n_{lt} \leq C_t \quad \forall t \tag{9}
\]

In addition, the value of \( n_{lt} \) is constrained to be zero if liquid \( l \) is not produced in week \( t \) or to between 1 and \( C_t \) if it is:

\[
z_{lt} \leq n_{lt} \leq C_t z_{lt} \quad \forall l, t \tag{10}
\]

The complete planning model is given by expressions (1) to (5) and (7) to (10). It is this formulation, denoted CLP (Canned Liquid Planning), that must be solved quickly so as to be suitable for operational use with frequent replanning.

3 Algorithmic solutions with only mathematical programming

Exact optimisation is illusory when production parameter values can vary over time and are not well known, demand forecasts are often revised, and upsets such as machine failure are common, necessitating frequent replanning. A more useful outcome may be a quickly-obtained plan of good quality. Bearing this in mind, a number of alternative solution approaches are now explored.

3.1 Default branch-&-bound search on the complete MIP model

The first approach is the ‘lazy’ default - simply let an industrial strength branch-&-bound MIP solver try to find a good solution within a short amount of time.

Branch-&-bound (BB) (Wolsey; 1998; Pinedo and Chao; 1999) is a well-established enumeration method for optimally solving MIPs, based on an upside-down tree search. For problems where the integer variables are only binary, each node in the tree has 2 branches descending from it, one for the value 0 of a binary variable, the other for the value 1, both leading to a node one level below. At each node, an integer feasible solution is sought, using a heuristic, in order to produce an upper bound on the node’s optimal solution and hopefully update the incumbent solution (the best integer feasible solution found so far at any node). If analysis shows that any solution subsequent to that node is either infeasible or has a lower bound that is higher than the incumbent, then the node is fathomed (i.e. eliminated) because it will not lead to an improved feasible solution. The optimal solution is the incumbent after all branches have been fathomed. If the branch and bound search has to be stopped prematurely, then the incumbent provides a best-so-far feasible solution whose distance from optimality is bounded by the smallest lower bound on any unfathomed node.

For problems with small numbers of integer variables, the BB approach is fast. However, as problem size grows, the number of different integer solution combinations explodes exponentially, possibly causing the search to take an impossibly long time to fathom all branches and converge to a provably optimal solution, as is shown to be the case for the problem of this paper.
The complete MIP model describes a big-bucket lot sizing problem with setup times (Belvaux and Wolsey; 1998), a class which is generally not trivial to solve. For the real 13-week data set used in this paper, the model had 715 binary variables ($y$ and $z$), 182 non-binary integer variables ($n$), 1781 continuous variables ($x, I^+, I^-$ and $u$), 2171 constraints and 7770 non-zero matrix elements after the presolve had eliminated 82 variables, so it is a sizeable problem, although not huge. It is, however, large enough not to be optimally solvable in a reasonable amount of computer time. The feasible integer solutions shown in Table 1 are the incumbent solution after increasing amounts of BB search time on a Sun Enterprise 450 workstation with a 400 MHz Ultrasparc CPU and 510 MB of RAM. The incumbent solution when the BB search ran out of memory after 13 hours was 395,823. Note that the lower bound in Table 1 is stationary after one minute at 339,528, about 15% below the final incumbent solution. Observe too that the solution does not improve much after the first 30 minutes of the search.

<table>
<thead>
<tr>
<th>Search Time</th>
<th>Best Solution (Incumbent)</th>
<th>Best Lower Bound</th>
<th>Branch-&amp;-Bound Nodes Searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 min</td>
<td>481,713</td>
<td>339,528</td>
<td>2,566</td>
</tr>
<tr>
<td>2 mins</td>
<td>481,713</td>
<td>339,528</td>
<td>5,333</td>
</tr>
<tr>
<td>5 mins</td>
<td>481,713</td>
<td>ditto</td>
<td>12,993</td>
</tr>
<tr>
<td>10 mins</td>
<td>481,713</td>
<td>...</td>
<td>26,429</td>
</tr>
<tr>
<td>15 mins</td>
<td>481,713</td>
<td>...</td>
<td>38,661</td>
</tr>
<tr>
<td>20 mins</td>
<td>430,532</td>
<td>...</td>
<td>51,600</td>
</tr>
<tr>
<td>30 mins</td>
<td>414,358</td>
<td>...</td>
<td>76,024</td>
</tr>
<tr>
<td>1 hour</td>
<td>414,358</td>
<td>...</td>
<td>136,471</td>
</tr>
<tr>
<td>1.5 hours</td>
<td>400,408</td>
<td>...</td>
<td>201,930</td>
</tr>
<tr>
<td>5 hours</td>
<td>400,408</td>
<td>...</td>
<td>616,063</td>
</tr>
<tr>
<td>10 hours</td>
<td>395,823</td>
<td>...</td>
<td>1,198,559</td>
</tr>
<tr>
<td>Out of memory</td>
<td>395,823</td>
<td>339,528</td>
<td>1,527,170</td>
</tr>
</tbody>
</table>

Table 1: Branch-&-bound results for the complete model

Clearly the identification of a provably optimal solution plan takes an impracticable amount of computer time and memory, motivating the development of the alternative approaches below.

### 3.2 Accelerated branch-&-bound search on the complete MIP model

An alternative to searching for a provably optimal solution in a BB search is to ruthlessly prune the tree by fathoming a node if its lower bound is greater than the incumbent solution value minus a certain proportion $\phi$ of the node’s upper bound. The BB default corresponds to $\phi = 0$. A positive value of $\phi$ will generally accelerate the search, but may miss the optimal solution by pruning a branch leading to it. Table 2 show the results of computational tests with a range of values of $\phi$ and a time limit of 1 hour imposed on the BB search. Note that the good results obtained for values of $\phi$ between 0.10 and 0.21, and the excellent solution of 367,089 (all stock, no backlogs) found in 720 seconds (12 minutes) when $\phi = 0.21$. However, the value of $\phi$ only needs to be increased slightly to 0.22 for the solution to deteriorate to 466,917 after an hour’s BB search.
### Table 2: Accelerated branch-&-bound results for the complete model

<table>
<thead>
<tr>
<th>Value of $\phi$</th>
<th>BB Solution</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>414,358</td>
<td>1 hour</td>
</tr>
<tr>
<td>0.05</td>
<td>385,002</td>
<td>1 hour</td>
</tr>
<tr>
<td>0.10</td>
<td>384,534</td>
<td>300 secs</td>
</tr>
<tr>
<td>0.15</td>
<td>392,153</td>
<td>460 secs</td>
</tr>
<tr>
<td>0.20</td>
<td>373,452</td>
<td>370 secs</td>
</tr>
<tr>
<td>0.21</td>
<td>367,089</td>
<td>720 secs</td>
</tr>
<tr>
<td>0.22</td>
<td>466,917</td>
<td>370 secs</td>
</tr>
<tr>
<td>0.25</td>
<td>466,917</td>
<td>720 secs</td>
</tr>
<tr>
<td>0.30</td>
<td>466,917</td>
<td>17 secs</td>
</tr>
<tr>
<td>0.35</td>
<td>466,917</td>
<td>17 secs</td>
</tr>
<tr>
<td>0.40</td>
<td>466,917</td>
<td>17 secs</td>
</tr>
</tbody>
</table>

This approach provides a useful benchmark as it identifies the best solution (367,089) found in the whole of this paper, but it is too unpredictable in terms of solution quality and computing time to be practical for operational use.

In an attempt to obtain an even better benchmark, the BB search with $\phi = 0$ was repeated with the 367,089 solution specified at the start as the incumbent, but no better solution was identified before the search ran out of memory after 14 hours.

### 3.3 Backward-then-forward pass

The third approach first optimizes production one period at a time in a backward pass through periods $T$, $T-1$, ..., 1 in order to identify the target stock levels $S_{pt}$ that would be necessary to fulfill demand after period $t$:

**Backward pass:**

- Take out the backorder variables $I^{-}_{pt}$ from formulation CLP.
- Let $S_{pT} = 0 \forall p$.
- For $t = T$ down to 1 do {
  - Fix $I^{-}_{pt} = S_{pt}$ as data $\forall p$, and free $I^{+}_{p,t-1}$ as variables $\forall p$.
  - Solve formulation CLP just for the $I^{+}_{p,t-1}$ and all period $t$ variables.
  - Let $S_{pt-1} = I^{+}_{p,t-1} \forall p$.
} (end of $t$ loop).

The target stock levels $S_{p0}$ for period zero provide indicative minimum levels for current stocks $I^{+}_{p0}$ that are needed to satisfy demand over the $T$-week planning horizon without backlogs. If effective demand is used (so that current stocks and backorders are zeroed), then an indicator of insufficient capacity to meet overall demand is the value $\sum_{p} h_{p} S_{p0}$.

After the backward pass above, a forward pass through periods 1, 2, ..., $T$ then optimizes production one period at a time in order to build up the target stocks levels $S_{pt}$.

**Forward pass:**

- For $t = 1$ to $T$ do {
  - Inflated demand by the target stocks levels: Let $d_{pt} = d_{pt} + S_{pt} \forall p$.
  - Solve formulation CLP for just the period $t$ variables.
  - Restore the original demand: Let $d_{pt} = d_{pt} - S_{pt} \forall p$.
}
Recalculate and then fix stocks $I_{p,t-1}^+$ and backorders $I_{p,t-1}^-$ as data $\forall \ p$

} (end of $t$ loop).

The above backward-forward approach is surprisingly effective giving a solution of 504,305 (all stock with zero backlogs) in just 2 seconds. This fast and practicable computing time includes the solving of $2T = 26$ MIPs, most with 55 binary variables each. If the targets stocks levels $S_{pt}$ identified in the backward pass are ignored (i.e., set to zero) then, not surprisingly, a far worse solution of 21,948,200 is obtained (with 73,270 stocks and 218,749 backlogs).

### 3.4 Forward pass with linear setup approximations

A fourth approach involves the elimination of the binary $y$ and $z$ variables representing the setups for week 2 onwards, compensating by increasing the values of the unit production times $a$ in those weeks.

The resulting model, denoted $CLP^*$, had just 55 binary variables, 14 integer variables, 1781 continuous variables, 851 constraints, 4302 non-zero matrix elements and was solved in under a second, i.e. very quickly. After solving for the setups of week 1, the model was reformulated so that the former week 2 is now week 1, and number of weeks in the model reduced by one from $T$ to $T-1$. The model was solved afresh, and so on, with $T-2$ further models being solved, until the setups for periods 1, ..., $T$ had all been decided. The whole series of models was solved in between 9 and 12 seconds, but the quality of the resulting solution depended on by how much the value of the unit production time $a$ was increased, as the tests below show.

In model $CLP^*$, constraints (3) - (5) and (7) - (10) all applied only to week 1. For weeks 2 to $T$, the integer tank $n$ variables were made continuous, and so the tank under-utilization $u$ variables were eliminated. Thus the constraints for week 2 onwards were:

\[
I_{p,t-1}^+ - I_{p,t-1}^- + x_{pt} - I_{pt}^+ + I_{pt}^- = d_{pt} \quad \text{(unchanged)} \quad \forall \ p, \ t \geq 2 \quad (11)
\]

\[
\sum_p a^* x_{pt} \leq B_t \quad \forall \ t \geq 2 \quad (12)
\]

\[
r_{lp} \sum_{p \in P(l)} x_{pt} = F_{n_{lt}} \quad \forall \ l, t \geq 2 \quad (13)
\]

\[
\sum_l n_{lt} \leq C_t \quad \forall \ t \geq 2 \quad (14)
\]

where $a^*$ is the unit production time with setups factored in.

The resulting provisional lot-sizes for weeks 2 to $T$ are never actually implemented, but merely used to approximately indicate future production which is thus taken account of in deciding week 1’s lot sizes and sequence. The actual lot sizes and sequence in each of weeks 2 to $T$ are all eventually determined via the week 1 constraints (3) - (5) and (7) - (10) of model $CLP$, hence ensuring capacity feasibility of production in all of weeks 1 to $T$.

By how much should $a$ be increased to $a^*$ ? From constraint (3), the value of $a^*$ has to satisfy

\[
\sum_p a^* x_{pt} = \sum_p a x_{pt} + s \left( \sum_p y_{pt} - 1 \right) + e \left( \sum_l z_{lt} - 1 \right) \quad \forall t \quad (15)
\]

Now, assuming that
1. the total production $\sum_t x_{pt}$ of a product $p$ equals its total effective demand $\sum_t d_{pt}$, which is a reasonable assumption unless overall demand far outstrips capacity

2. the canning line is using all available capacity time on setups and production, so that $B_t$ is equal to the right hand side of constraint (3) and hence, by (15), to $\sum_p a^* x_{pt}$ then $\sum_{p,t} a^* d_{pt} = \sum_t B_t$ and so the following expression for the inflated unit production time $a^*$ can be derived

$$a^* = \frac{\sum_t B_t}{\sum_{p,t} d_{pt}} \quad \forall \ p$$ (16)

resulting, for the data in hand, in more than a 54.5% increase from $a$ to $a^*$.

Using $a^* = 1.545a$ does indeed a reasonable result of 628,910 (in 9 seconds), but it is still inferior to that of 504,305 (with zero backlogs) in just 2 seconds of computing time resulting from the backward-then-forward pass of Section 3.3. Are there values of $a^*$ that produce a better solution? In Figure 1 (at the end of this paper, with all other figures) the top graph (a) shows how the solution varies for values of $a^*$ over intervals of $10^{-3}a$ between $0.5a$ and $2.5a$. It can be seen that the minimum solution is around 400,000 and lies somewhere near 1.2a. The graph also illustrates very clearly that not factoring in setup times, i.e. using $a^* = 1.0a$, results in a solution of about 970,000, well over 100% above the minimum. Note how the solution suddenly falls sharply from 968,823 at 1.189a to 477,860 at 1.201a. Graph (b) in Figure 1 shows the huge reduction in backlogs, almost to zero, that is achieved by the value of $a^*$ being sufficiently large to force peaks of demand to be produced earlier in time. Note also the subsequent and gradual rise of the solution and backlog values from above 1.5a as $a^*$ becomes too large, causing premature preproduction of peaks of demand. It took 5 hours and 40 minutes to calculate all 2000 plotted solutions in Figure 1, hardly an efficient way of finding a near-optimal solution.

A quicker and more effective method would be to perform an intelligent search for the optimal value of $a^*$. This is easier said than done, as can be seen from the top graph (a) of figure 2 which shows how the solution varies in a spiky manner for values of $a^*$ over intervals of $10^{-4}a$ between 1.2100a and 1.2600a. The bottom graph (b) reinforces the impression of spikiness of solution values over intervals of $10^{-5}a$ between 1.2320a and 1.2350a. In other words, the solution is sensitive to small changes in the value of $a^*$ which are altering the combinatorial schedule of setups which in turn impact on the availability of canning line time and hence on the stocks and, to a lesser extent in this range, the backlogs. For example, the three values 1.23256a, 1.23257a, and 1.23258a have respectively the solutions 413,571, 402,87 and 423,401, all three of which differ between eachother with respect to several of the tank variables $z_{lt}$ (although all three have just 17 units of backlogs). As can be seen on graph (b) of figure 2, the third solution value 423,401 occurs frequently over the range 1.232a to 1.235 (and beyond). Inspection of these solutions reveals that they differ also slightly with respect to the tank variables $z_{lt}$, but again all have just 17 units of backlogs as, in fact, do all solutions from 1.21875 to 1.26.

This spikiness has implications for the way we try to automatically identify the best value of $a^*$ for a given problem instance, as will be discussed in the next section.

4 Hybrid heuristics with local search and mathematical programming

The final two approaches both use neighbourhoud or local search (Pirlot; 1996; Aarts and Lenstra; 1997; Sait and Youssef; 1999), a metaheuristic iterative method for finding an
optimal or good solution to general combinatorial problems. In the first iteration, a starting solution is identified and adopted as the current solution. At each iteration, the current starting solution is adjusted to produce one or more so-called neighbouring solutions. In the most simple form of local search, known as random hill climbing or ascent algorithm, if a randomly selected neighbouring solution has an objective function that is an improvement over the current solution, then that neighbour is adopted as the new current solution. If not, then further randomly selected neighbouring solutions are evaluated until an improved one is found. If it is not possible to find a better solution within a certain amount of time or effort, then the local search is stopped and the current solution hailed as the best one found.

The simple form of local search can be likened, in the case of a maximization problem, to a person trying to walk up a hill blindfolded by feeling around with your feet for a direction with an upward slope and then walking one or more steps in that direction. At the new position, the person feels around again for an upwards slope. In this manner, she or he will eventually reach the top of the local high point, but probably will not reach the top of the highest point in the country, let alone Mount Everest. Thus the simple ascent form of local search will get near a local optimum, but not necessarily near the global optimum, depending on the complexity of the problem, how neighbouring solutions are defined, and the computing effort one is prepared to spend on the search.

Various strategies have been proposed to encourage the local search to not get stuck at a local optimum, but rather to converge to the global optimum, such as simulated annealing (Sait and Youssef; 1999; Dowsland; 1993; Aarts et al.; 1997), which permits moving to inferior neighbouring solutions with decreasing probability, the related strategy of threshold accepting (Dueck and Scheuer; 1990; Lin et al.; 1995; Glass and Potts; 1996; Meyr; 2000) where inferior moves are accepted deterministically if they worsen by no more than a decreasing threshold, and tabu search (Sait and Youssef; 1999; Glover and Laguna; 1997; Hertz et al.; 1997) where recent moves are banned through a tabu list of temporarily prohibited moves. A good tutorial and comparative discussion of both these and other local search methods can be found in Pirlot (1996).

The local search approach proposed below starts with a very wide range of neighbours in order to generate a diverse range of solutions and avoid premature entrapment at a local optimum. The size of the neighbourhood gradually diminishes as the search progresses in order to home in on a good solution. Moves to an inferior solution are not permitted.

The challenge here is to identify the size of an effective neighbourhood structure at different stages of the search. At one extreme, a small neighbourhood structure could mean a very slow convergence rate, coupled with the risk of getting trapped in a local optimum. At the other extreme, a very wide ranging structure would lean in the direction of pure random sampling among all possible solutions. Random sampling has the advantage that it would avoid entrapment in a local optimum, but is almost certainly inefficient at identifying good solutions by nature of its unguided randomness. A possible compromise is to start a local search with the largest possible neighbourhood to avoid bad or mediocre local optima, and then narrow the neighbourhood as the search progresses in order to home in to a good solution, even if it is still a non-global optimum. The approach is given the name diminishing-neighbourhood search. Other authors (Reeves; 1994; Aluja et al.; 1999; Mühlenbein and Zimmerman; 2000; Hansen and Mladenovic; 2001) have advocated the use of large or varying-sized neighbourhoods to avoid local optima, but not in the continuously diminishing manner proposed here.
4.1 Local Search on the forward pass with linear setup approximations

The spikiness manifested in Figures 1 and 2 suggests that some neighbourhood searches on $a^*$ might get stuck in a local optimum far from the global optimum around $1.2a$. How then does diminishing neighbourhood search perform? Computational tests were performed using the following search approach:

1. Start with a reasonable estimate of $a^*$, such as the value of 1.545 from expression (16). Make this the current solution.
2. Set the neighbourhood radius $N$ to an initial value $IN$.
3. Iterate $N$ times. At each iteration:
   (a) Generate a neighbouring solution by randomly selecting a real number $r$ in the interval $[-N, +N]$ and setting neighbouring $a^* = \text{current } a^* + r$.
   (b) Solve model $CLP^*$ using neighbouring $a^*$.
   (c) Move to the neighbouring $a^*$ solution if it is better than the current one.
   (d) Reduce $N$ by multiplying by a constant factor $f$.
4. Stop and output the current solution as the best found by the search.

Starting with a good estimated solution (here 1.545), the starting neighbourhood radius $IN$ should be set sufficiently large to be able to encompass the likely optimal solution, although this is not strictly necessary and will sometimes be difficult to judge in practice.

In general, a parameter determining the effectiveness of diminishing neighbourhood search will be the rate at which the neighbourhood size diminishes. If this rate is too small for the number of iterations carried out, then too much time will be spent on random-like search (diversification) and not converge sufficiently (intensification) to a good solution. If the rate is too large, then too much intensification will occur in relation to diversification and the search will prematurely home in to a local optimum which may be much worse than the global optimum. In this application, $N$ is reduced by being multiplied by a constant factor $f < 1$ at every iteration.

Computational tests were carried out using $N = 100$ iterations, each search taking about 17 minutes. Using principally an initial neighbourhood radius $IN = 0.5$, alternative reduction factors $f = (FN/IN)^{1/N}$ were each tested 50 times, corresponding to a number of finishing radiuses $FN$, as well as a random search over $1.545 \pm IN$ for benchmarking purposes. Note that $IN = FN$ implies a pure descent algorithm.

The results are shown in table 3. The parameters $IN$ and $FN$ were initially estimated as 0.5 and 0.0005 respectively and indeed these performed best, giving a solution of 408,965. Using $N = 1000$ iterations with the same initial and final radiuses improved slightly on this, providing a solution of 404,549. Several other initial radiuses with $FN = 0.0005$ were also tested over 100 iterations, but with inferior results. The pure descent algorithm and random search performed poorly, indicating the power of diminishing-neighbourhood search to cope with the spikiness of the $a^*$ landscape if appropriate initial and final radiuses are used. It is interesting that an initial radius of $IN = 1.0$ produced a solution only slightly poorer than that for $IN = 0.5$, possibly suggesting that it is safer to err on the side of a larger initial neighbourhood than a smaller one.
<table>
<thead>
<tr>
<th>No. of</th>
<th>Starting Radius N</th>
<th>Finishing Radius N</th>
<th>No. of Tests</th>
<th>Mean Solution</th>
<th>Best Solution</th>
<th>$a^*$ of Best Solution</th>
<th>Mean Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>0.0005</td>
<td>50</td>
<td>410,326</td>
<td>402,777</td>
<td>1.233527</td>
<td>16.7 m</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
<td>Random S</td>
<td>50</td>
<td>418,833</td>
<td>410,687</td>
<td>1.234001</td>
<td>17.4 m</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
<td>0.5</td>
<td>50</td>
<td>419,812</td>
<td>402,512</td>
<td>1.225303</td>
<td>16.5 m</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
<td>0.05</td>
<td>50</td>
<td>415,083</td>
<td>402,628</td>
<td>1.233635</td>
<td>16.6 m</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
<td>0.005</td>
<td>50</td>
<td>411,458</td>
<td>404,577</td>
<td>1.231682</td>
<td>16.5 m</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
<td>0.0005</td>
<td>50</td>
<td>410,296</td>
<td>402,857</td>
<td>1.232674</td>
<td>16.5 m</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>0.2</td>
<td>50</td>
<td>415,946</td>
<td>408,753</td>
<td>1.222975</td>
<td>16.4 m</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>0.0005</td>
<td>50</td>
<td>434,083</td>
<td>402,628</td>
<td>1.233667</td>
<td>16.8 m</td>
</tr>
<tr>
<td>100</td>
<td>0.1</td>
<td>0.0005</td>
<td>50</td>
<td>449,073</td>
<td>405,624</td>
<td>1.232552</td>
<td>17.1 m</td>
</tr>
<tr>
<td>1000</td>
<td>0.5</td>
<td>0.0005</td>
<td>20</td>
<td>404,549</td>
<td>400,390</td>
<td>1.228465</td>
<td>168.6 m</td>
</tr>
</tbody>
</table>

Table 3: Diminishing neighbourhood and descent algorithm on the forward pass with linear setup approximations

4.2 Local search on the binary tank variables

Diminishing-neighbourhood and the descent algorithm are now applied to the binary tank variables $z_{lt}$ of model $CLP$. A neighbouring solution is generated by carrying out a large number of random changes to the 0/1 values of the $z$ variable of the current solution. As the search progresses, the number of such changes needed to generate a neighbour is slowly reduced. At each iteration of the search, a reduced-size MIP is solved, not optimally, but only approximately in order to rapidly obtain a measure of the quality of the solution being evaluated. This enables the search to proceed and identify good solutions relatively quickly.

Specifically, the procedure is:

1. Start with the incumbent solution $\{z_{lt} \mid \forall l, t\}$ after one minute of BB search.
2. Set $N = LT = 182$, the number of $z_{lt}$ variables.
3. Iterate $N$ times. At each iteration:
   
   (a) Generate a neighbouring solution by randomly selecting $N$ distinct pairs $(l, t)$ and then randomly deciding if each pair should be toggled by letting $z_{lt} = 1 - z_{lt}$ with probability 0.5. Ensure that at least one randomly chosen pair is toggled. Fix the values of the $z_{lt}$ variables.
   
   (b) Solve the model for the integer $n_{lt}$, binary $y_{pt}$, and continuous $u_{lt}$, $x_{pt}$, $I^+_p$, & $I^-_p$ variables, using BB search, stopping at the first integer solution.
   
   (c) Move to the neighbouring solution if it is better than the current one.
   
   (d) Reduce $N$ (as discussed below).
4. Stop and output the current solution as the best found by the search.
5. Fix the values of the $z_{lt}$ variables in the best solution in step 4 and carry out a BB search over the remaining variables for five minutes.
As there are $LT\ z_l$ variables, all solutions are reachable and equally probable in step 3a when the search is started with an initial value of $N = LT$ (as in step 2). Generally, for any value of $N$, all solutions that differ by $N$ or fewer 0/1 values are reachable at step 3a with equal probability.

For the canning line data of this paper, it takes between 0.15 and 0.7 seconds in step 3b to find the first integer feasible solution. This is a more viable time at each iteration of the search than spending even a minute looking for a better BB solution. However, it does mean that the solution value is generally an approximation to the optimal value at that iteration, inevitably blunting the search, but still effective in that the time saved will permit a vastly greater number of iterations to be carried out.

A parameter determining the effectiveness of the search will be the rate at which the neighbourhood diminishes, i.e. the rate of reduction of $N$ in step 3d. A real-valued parameter $RealN$, initially set to $N$, is reduced by being multiplied by a constant factor $f < 1$ at every iteration and then used to let $N = \text{ceiling}(RealN)$, the integer at or immediately above $RealN$. If the value of $f$ is too small for the number of iterations carried out, then the search will spend too much time on random-like search (diversification) and not converge sufficiently (intensification) to a good solution. If $f$ is too large, then too much intensification will occur in relation to diversification and the search will prematurely home in to a local optimum which may be much worse than the global optimum. Rather than specify the values of $f$ in the computational tests below, $f$ was calculated as

$$f = (LT)^{-\frac{1}{qN}}$$  \hspace{1cm} (17)

where $N$ is the number of iterations in the local search and $q$ the percentage of the search after which the smallest neighbourhood of size $N = 1$ was reached in step 3, as illustrated in Figure 3.

Computational experiments were carried out to compare the method with the other approaches. The number $N$ of iterations per search was $N = 1000$ and 10,000. The parameter varied was the percentage $q$ of the search after which the smallest neighbourhood of size $N = 1$ was reached, using test values $q = 0\%, 25\%, 50\%, 75\%, 100\%$, as shown in parts (a)-(e) of Figure 3. In addition, for comparison, the tests include the curve (with $f = (0.5)^{1/N}$) for which $N = LT/2$ at the end of the search as in part (f) of Figure 3.

1000 iterations of local search took approximately 10 minutes while 10,000 iterations took about ten times as long. At the end of the local search, a further 5 minutes was spent on a final BB search in a generally successful attempt to improve the solution. Twenty-five 1000-iteration searches were carried out for each of the 6 curves, but due to the larger computing times, only ten 10,000-iteration searches were made for the 6 curves. The results are shown in Table 4.

For both 1000 and 10,000 iterations, the diminishing neighbourhood method gives its best mean solutions values of $q$ between 25% and 100% and appears to be robust within this range. The improvement from 1000 to 10,000 iterations is clear. The curve where $N = LT/2$ at the end of the search results in absolutely no improvement during the search - the solution of 427,088 is due to the the final 5 minutes of BB search improving on the initial solution of 481,713, after fixing the $z_l$ variables. When 1000 iterations are used, the curve $q = 0\%$ (i.e. a simple local search with no diversification at all, as the smallest neighbourhood of $N = 1$ is used throughout) results in a mean solution that is a little better than those for curves $q = 25\%$ to 100%.

However, when 10,000 iterations are used, the mean solution for $q = 0\%$ only slightly improves to around 411,271, resulting in a worse mean solution than for $q = 25\%$ to 100%.
1000 iterations  
25 runs (16 m each)  
<table>
<thead>
<tr>
<th>Curve: Rate of Reduction of N</th>
<th>Mean Solution</th>
<th>Best Solution</th>
<th>Mean Solution</th>
<th>Best Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descent: $q = 0%$</td>
<td>411,855</td>
<td>383,592</td>
<td>411,271</td>
<td>387,743</td>
</tr>
<tr>
<td>DNS: $q = 25%$</td>
<td>414,107</td>
<td>384,230</td>
<td>401,738</td>
<td>376,952</td>
</tr>
<tr>
<td>DNS: $q = 50%$</td>
<td>414,739</td>
<td>388,101</td>
<td>394,006</td>
<td>379,325</td>
</tr>
<tr>
<td>DNS: $q = 75%$</td>
<td>413,199</td>
<td>393,026</td>
<td>402,711</td>
<td>380,977</td>
</tr>
<tr>
<td>DNS: $q = 100%$</td>
<td>416,259</td>
<td>376,287</td>
<td>399,365</td>
<td>388,137</td>
</tr>
<tr>
<td>DNS: $N \gg LT/2$</td>
<td>427,088</td>
<td>427,088</td>
<td>427,088</td>
<td>427,088</td>
</tr>
</tbody>
</table>

Table 4: Diminishing neighbourhood search results

This suggests that the descent algorithm tends to get trapped near a local optimum at between 1000 and 10,000 iterations, as evidenced by the fact that the best solution in a 10,000-iteration search was reached after just 2811 iterations on average. Even the $q = 25\%$ curve, with its limited diversification at the start, enables the diminishing neighbourhood search to avoid mediocre local optima and progress by iteration 3242 on average to a better mean solution of 401,738 (whose constituent solutions are almost certainly local optima, given that the remaining average of 6758 iterations did not identify a better solution from the $LT = 182$ neighbours within the $N = 1$ neighbourhood).

For curves $q = 25\%$ to $100\%$, the final five minutes of BB search in step 5 make a mean difference of about $2\frac{1}{2}\%$ for 1000 iterations (improving the mean solution from about 425,000 to around 414,500), and of about $1\frac{1}{2}\%$ for 10,000 iterations (improving the mean solution from 405,300 to 399,500). This suggests that the final five-minute BB search is worthwhile, but not crucial. The gains in the final BB search indicate that some of the improvement potential lost in step 3b by terminating the BB search at the first integer solution is recoverable in step 5. It would be totally recoverable if in step 3b the rank ordering of 5-minute solutions and of first integer solutions were the same, thus following identical search trajectories. From the perspective of optimality, the implicit assumption in step 5 that the rankings of the first integer solutions and optimal solutions are positively correlated is not unreasonable and, while blunting the search, certainly makes it viable in reasonable computing time.

5 Discussion and conclusions

Table 5 summarises the mean quality and computing time of the solutions obtainable in the operational use of the heuristics. The results of Table 2 are thus excluded from consideration because they offer no guidance as to what is a good (let alone best) value of $\phi$ that can expected $a$ priori to give good solutions. The value of $\phi$ has only to shift slightly from 0.21 to 0.22 for the solution to change from the best in this paper (367,089 in 720 seconds) to a mediocre one (466,917 in 1 hour). In Tables 3 and 4, diminishing-neighbourhood (DN) search showed itself to be more robust with respect to its parameters.

Note that all but the first of the branch-&-bound (BB) solutions of table 5 are dominated by at least one other solution which took up to the same time in providing a solution of at least the same quality. Figure 4 is a scatter graph of the solutions that took up to 110 minutes and shows the efficient frontier (Goodwin and Wright; 1998) that links all
Table 5: Summary of heuristic solutions and computing times (in decreasing order of solution value)

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Mean Solution</th>
<th>Mean Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 3.3: Backward-then-forward pass</td>
<td>505,000</td>
<td>0.03 min</td>
</tr>
<tr>
<td>Table 1: Default BB search for 1 minute</td>
<td>481,713</td>
<td>1 min</td>
</tr>
<tr>
<td>Table 1: Default BB search for 20 minutes</td>
<td>430,532</td>
<td>20 min</td>
</tr>
<tr>
<td>Table 3: 100 iterations of descent algorithm for (a^*)</td>
<td>419,812</td>
<td>16.5 min</td>
</tr>
<tr>
<td>Table 1: Default BB search for 30 minutes</td>
<td>414,358</td>
<td>30 min</td>
</tr>
<tr>
<td>Table 4: 1000 iterations of DN search on (z_{lt}) (av.)</td>
<td>414,000</td>
<td>16 min</td>
</tr>
<tr>
<td>Table 4: 1000 iterations of descent algorithm for (z_{lt})</td>
<td>411,855</td>
<td>16 min</td>
</tr>
<tr>
<td>Table 4: 10,000 iterations of descent algorithm for (z_{lt})</td>
<td>411,271</td>
<td>110 min</td>
</tr>
<tr>
<td>Table 3: 100 iterations of DN search on (a^*)</td>
<td>408,965</td>
<td>16.5 min</td>
</tr>
<tr>
<td>Table 1: Default BB search for 90 minutes</td>
<td>400,408</td>
<td>90 min</td>
</tr>
<tr>
<td>Table 1: Default BB search for 300 minutes</td>
<td>400,408</td>
<td>300 min</td>
</tr>
<tr>
<td>Table 3: 1000 iterations of DN search on (a^*)</td>
<td>404,549</td>
<td>169 min</td>
</tr>
<tr>
<td>Table 4: 10,000 iteration of DN search on (z_{lt})</td>
<td>394,006</td>
<td>110 min</td>
</tr>
</tbody>
</table>

The non-dominated solutions, providing a user with several alternatives depending on how much time is available to obtain a result. If an instant solution is needed, then use the forward-then-backward pass of Section 3.3. If just a few minutes are available, then default branch-&-bound will suffice. The 120° lower left corner of the efficient frontier wraps a tight cluster of three nearly equal solutions obtainable after 16 or 17 minutes with DN search on \(a^*\) performing best. These are the ‘best value for money’ solutions in that only slightly better BB and DNS solutions are obtainable with much more computing time.

The \(a^*\) approach performs well with diminishing-neighbourhood search. However, further research is needed to investigate its behaviour and performance on more complex problems, such as multiple non-identical machines where the unit production time \(a\) varies with product and machine. In this case, the local search becomes more challenging. Diminishing-neighbourhood search and the descent algorithm on the tank binary variables \(z_{lt}\) also perform well, with the descent algorithm tending to get stuck in a local optimum after 3000 iterations.

The research of this paper was carried out after the termination of the project at the drinks manufacturer, but the model and data are typical of many production planning challenges encountered in practice. The purpose of the paper submission was to inform both the academic and practitioner industrial engineering community of the interesting and useful results. These indicate that hybrid heuristics mixing both classical branch-&-bound methods and modern local search approaches are worth exploring to accelerate a search for good solutions in industrial production planning and scheduling.
References


© 2003 Alistair Clark, All Rights Reserved
Figure 1: Solution (a) and backlog (b) values as a function of $a^*$
Figure 2: Solution value as a function of $a^*$
Figure 3: Reduction rate of the neighbourhood radius during the search

(a) $q = 0\%$

(b) $q = 25\%$

(c) $q = 50\%$

(d) $q = 75\%$

(e) $q = 100\%$

(f) $N = LT/2$ at end of search
Figure 4: Efficient frontier of the solutions and times of Table 5