

## Template Design under Demand Uncertainty by Integer Linear Local Search

S. D. PRESTWICH†\* , S. A. TARIM† and B. HNICH‡

(Received July 2005; In final form December 2005)

Production planning under uncertain demands leads to optimisation problems that are hard both to model and to solve. We describe an integer linear model for a template design problem under uncertainty, and investigate its solution by a general-purpose local search algorithm for integer linear programs. Several such algorithms have previously been proposed as tools for solving large combinatorial optimisation problems, and ours is based on a recent Boolean Satisfiability algorithm. In experiments it was slower than other methods on small instances, but rapidly outstripped them as the problem size and number of templates increased. It also found near-optimal solutions to all instances much more quickly. A general-purpose local search algorithm provides a rapid and convenient way of finding high-quality solutions to complex production problems.

### 1 Introduction

The *probabilistic template design problem* is a practically-motivated production planning problem that is combinatorial in nature and has inherent demand uncertainty. A range of techniques from Operations Research and Artificial Intelligence — such as mathematical programming and constraint programming — have been applied to such problems, but these do not always scale up to large instances. Stochastic techniques such as evolutionary search are often more effective on large optimisation problems. Though they are not guaranteed to find optimal solutions, nor to prove their optimality, they provide a pragmatic way of quickly finding high-quality solutions. Evolutionary search is also very convenient when one is faced with new problems: a general-purpose evolutionary algorithm may be re-used many times, and one need only find an effective representation for any new problem. Modelling new problems is certainly non-trivial, but much easier than developing a new algorithm. In this sense, evolutionary search is more convenient than traditional heuristic

---

‡ Faculty of Computer Science, Izmir University of Economics, Izmir, Turkey

† Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland

\*To whom correspondence should be addressed. e-mail: s.prestwich@cs.ucc.ie

search approaches, in which one implements a new algorithm from scratch by defining local moves and local neighborhoods.

For example, one of the earliest heuristic search algorithms was for the Travelling Salesman Problem (TSP) (Lin and Kernighan (1973)). Each search state is a Hamiltonian Tour of the cities, and local moves permute a small number (usually two or three) of the cities to try to reduce the total distance travelled. This algorithm gives very good results but is specialised to the TSP. For more complex problems it is much easier to model them using a more general formalism such as integer programs, which can be solved by an off-the-shelf branch-and-bound algorithm (see Ormond and Williams (2004) on integer models for the asymmetric TSP, for example). Similarly, Boolean Satisfiability (SAT) problems are usually modelled in a standard language, then solved by open-source search algorithms based on local search or backtracking.

We propose a similar approach to production planning and related problems: modelling new problems in a simple language, then applying a fixed local search algorithm for that language. We evaluate this approach on the problem of template design under uncertain demand. The algorithm is not tailored to the problem, but is a general-purpose method for finding high-quality solutions to (full) integer programs. Nevertheless it gives good results on a set of benchmark problems. Section 2 describes the template design problem. In Section 3 a certainty-equivalent representation of the probabilistic problem is given via scenario analysis. A local search method for solving integer programs is presented in Section 4. In Section 5 the experimental results on the probabilistic template design problem are given. Section 6 concludes the paper and discusses future work.

## 2 The template design problem

First we describe the original deterministic problem, then elaborate it to probabilistic demands.

### 2.1 *Template design under deterministic demand*

The *template design problem* was first described by Proll and Smith (1998) who observed this problem arising at a local colour printing firm producing a variety of products from thin board, including cartons for human and animal food and magazine inserts.

The problem is described as follows. Given is a set of variations of a design, with a common shape and size and such that the number of required *pressings* of each variation is known. The problem is to design a set of templates, with a common capacity to which each must be filled, by assigning one or more

instances of a variation to each template. A design should be chosen that minimises the total number of *runs* of the templates required to satisfy the number of pressings required for each variation. As an example, the variations might be for cartons for different flavours of cat food, such as fish or chicken, where ten thousand fish cartons and twenty thousand chicken cartons need to be printed. The problem would then be to design a set of templates by assigning a number of fish and/or chicken designs to each template such that a minimal number of runs of the templates is required to print all thirty thousand cartons.

Proll and Smith address this problem by fixing the number of templates and minimising the total number of pressings. Let  $\mathcal{T}$  be the fixed-size set of templates, each with capacity  $s$ , to which variations are to be assigned. Let  $\mathcal{V}$  be the set of variations. Each variation  $i$  is described by a demand  $d_i$  that specifies the minimum number of pressings required. Proll and Smith's model comprises two sets of decision variables:  $R_i$  denotes the number of times template  $i$  is printed, and  $T_{i,j}$  the number  $k$  of instances assigned to template  $i$  of variation  $j$ , where  $0 \leq k \leq s$ . Every template has all its  $s$  slots occupied:

$$\forall i \in \mathcal{T} \cdot \sum_{j \in \mathcal{V}} T_{i,j} = s$$

The total production of each variation is no less than its demand:

$$\forall j \in \mathcal{V} \cdot \sum_{i \in \mathcal{T}} R_i T_{i,j} \geq d_j$$

The possible variable values are restricted by:

$$T_{i,j} \in \{0, \dots, s\}, \quad R_i \in I^{0,+}$$

The objective is to minimise the total number of pressings:

$$\sum_{i \in \mathcal{T}} R_i$$

## 2.2 Template design under probabilistic demand

In today's rapidly changing market environment, the deterministic demand assumption of the template design problem is no longer realistic, and the demand uncertainty must be addressed explicitly. Assuming that demand is uncertain, the previous model becomes inadequate because the model will

assume the worst case scenario for each demand. It will therefore return a solution with very high and unrealistic cost.

To overcome this problem, Tarim and Miguel (2005) propose a probabilistic model as follows. In this model  $\delta_j$  denotes random demand for variation  $j$  which follows a discrete probability distribution. In compliance with production/inventory theory, they incorporate two conventional cost components: scrap cost  $c_h$  and shortage cost  $c_p$ . The model has only one constraint that ensures that every template has all its  $s$  slots allocated:

$$\forall i \in \mathcal{T} \cdot \sum_{j \in \mathcal{V}} T_{i,j} = s$$

plus constraints restricting the variable values:

$$T_{i,j} \in \{0, \dots, s\}, \quad R_i \in I^{0,+}$$

A possible objective, however, is to minimise the expected total cost:

$$E \left( \sum_{j \in \mathcal{V}} c_p \min \left( 0, \sum_{i \in \mathcal{T}} R_i T_{i,j} - \delta_j \right) + \sum_{j \in \mathcal{V}} c_h \max \left( 0, \sum_{i \in \mathcal{T}} R_i T_{i,j} - \delta_j \right) \right)$$

### 3 Reformulating the probabilistic problem

The probabilistic model above has two major modelling bottlenecks: it contains random variables ( $\delta_j$ ) for expressing the probabilistic discrete demand, and it contains non-linear terms ( $R_i T_{i,j}$ ). Next we present a certainty-equivalent, but non-linear, model proposed in Tarim and Miguel (2005), then a new linear model.

#### 3.1 A certainty-equivalent model

We assume that the uncertain demand is expressed as a discrete probability distribution function, e.g.,  $\delta_1 = \{3(0.3), 7(0.5), 12(0.2)\}$ , where the values in parentheses represent the corresponding probability of a particular demand value. We also let  $\mathcal{S}$  denote the set of all possible scenarios of the demand process,  $p_i$  the probability of scenario  $i$ ,  $d_j^i$  the demand for variation  $j$  in scenario  $i$ ,  $e_j^i$  the excess quantity of variation  $j$  in scenario  $i$ ,  $b_j^i$  the amount of shortage of variation  $j$  in scenario  $i$ , and  $x_j$  be an auxiliary variable. The model

has the following constraints. Every template has all its  $s$  slots occupied:

$$\forall i \in \mathcal{T} \cdot \sum_{j \in \mathcal{V}} T_{i,j} = s$$

The total production of each variation  $j$  is equal to  $x_j$ :

$$\forall j \in \mathcal{V} \cdot \sum_{i \in \mathcal{T}} R_i T_{i,j} = x_j$$

The total production of each variation  $j$  in each scenario is the sum of demand plus the excess minus the shortage in that scenario:

$$\forall j \in \mathcal{V} \cdot \forall i \in \mathcal{S} \cdot x_j = d_j^i + e_j^i - b_j^i$$

The variable values are restricted by:

$$T_{i,j} \in \{0, \dots, s\}; \quad R_i \in I^{0,+}; \quad x_j, e_j^i, b_j^i \geq 0$$

The objective is to minimise:

$$\sum_{j \in \mathcal{V}} \sum_{i \in \mathcal{S}} p_i (c_p b_j^i + c_h e_j^i)$$

### 3.2 A linear model

From our point of view, the drawback of the previous model is its non-linearity. We desire a model that is both deterministic and linear, so that we can apply a local search algorithm for ILP. The similarity between the well-known cutting stock problem and the template design problem suggests that a linear formulation based upon the use of predetermined list of possible template designs is possible. However, this approach requires the generation of all feasible template designs, the number of which grows exponentially with the problem size. We present a new linear model that does not require the generation of all designs.

To avoid multiplying decision variables we define new 0/1 variables  $a_{ijk}$ , where  $a_{ijk} = 1$  if and only if variation  $j$  is placed in slot  $k$  of template  $i$ . Exactly one variation must be placed in each slot:

$$\sum_j a_{ijk} = 1$$

We also define integer variables  $f_{ijk}$  to model the contribution of slot  $k$  of template  $i$  to the total number of copies of variation  $j$ . If variation  $j$  is placed in slot  $k$  of template  $i$  then the contribution is  $R_i$ , otherwise it is zero ( $M$  is some large number):

$$f_{ijk} \leq R_i + M(1 - a_{ijk}) \quad R_i \leq f_{ijk} + M(1 - a_{ijk}) \quad f_{ijk} \leq Ma_{ijk}$$

The total number of variation  $j$  minus the demand is equal to the excess minus the backlog:

$$\sum_i \sum_k f_{ijk} - d_j^s = e_j^s - b_j^s$$

The variable values are restricted by:

$$a_{ijk} \in \{0, 1\}; \quad R_i \in I^{0,+}; \quad f_{ijk}, e_j^s, b_j^s \geq 0$$

The objective is to minimise:

$$\sum_j \sum_s (c_h e_j^s + c_p b_j^s)$$

This big-M formulation would be unsuitable for solution by branch-and-bound, because the use of large coefficients such as  $M$  weakens the linear relaxation, severely reducing the efficiency of branch-and-bound. This does not affect its usefulness for other solution methods, such as a constraint-based backtracking algorithm, but another feature does: a template design can now be represented in an exponential number of different ways because the placement of variations in a template can be permuted, so we have introduced a great deal of symmetry. Thus neither an integer programmer nor a constraint programmer would be likely to propose and test such a model (though additional constraints could be added to the model to remove the symmetries).

However, when modelling for a local search algorithm, quite different principles apply. A tight relaxation is unimportant, and previous work has shown that symmetry is harmless (Prestwich (2003)). In fact trying to remove the symmetry can be harmful, possibly because it reduces the relative size of the basins of attraction of solutions (Prestwich and Roli (2005)). For local search the size of the search space is less important than the number and distribution of solutions. There is therefore no reason to suppose that this model is unsuitable for local search.

#### 4 Local search for ILP

A few local search algorithms for forms of integer program have been reported. General purpose simulated annealing (GPSIMAN) (Connolly (1992)) is an early algorithm for 0/1 problems using Simulated Annealing (SA). From a feasible assignment a variable is selected and its value flipped (reassigned from 0 to 1 or vice-versa), then SA is used to restore feasibility. This approach was generalised to integer variables by (Abramson and Randall (1999)). Pedroso (1998) describes an evolutionary algorithm for MIP that searches on the integer variables, then uses linear relaxation to fix the continuous variables. A related approach used a version of the Greedy Randomised Adaptive Search Procedure (GRASP) (Neto and Pedroso (2004)) for MIP. GRASP is a meta-heuristic that alternates constructive phases to find good solutions with local search phases to find locally optimum solutions. TABU search has also been applied to MIP, for example see Lokketangen and Glover (1995). TABU is another meta-heuristic that prevents recent local moves from being reversed, to improve the likelihood of escaping from a local optimum.

A great deal of research has been done on local search for SAT (see Hoos and Stützle (2004) for a survey), which can be regarded as a restricted form of integer program. Modern local search algorithms for SAT routinely solve problems with hundreds of thousands of variables and millions of clauses (constraints on literals). An early but still successful algorithm is Walksat (McAllester *et al.* (1997), Selman *et al.* (1994)) which works as follows. Firstly, random truth values are assigned to all variables, so that each clause is either satisfied or violated. A violated clause is randomly selected; with some probability  $p$  a randomly chosen variable in the clause is flipped (has its value changed) in a *random walk move*; otherwise with probability  $1 - p$  a variable is heuristically selected and flipped in a *greedy move*; repeat until no violated clauses remain. Tabu lists and other techniques may be added. Though very simple, this combination of heuristics has performed well on many SAT benchmarks.

Unfortunately, on some problems Walksat can become trapped for long periods in deep local minima. To counteract this effect, *clause weighting* has been introduced. Clauses are dynamically weighted, and the search algorithm minimises the sum of the weights of the violated clauses. Clauses that are frequently violated are assigned larger weights. Weights may be updated additively or (borrowing a technique from machine learning) multiplicatively, and are periodically *smoothed* so that the search adapts to the current region of the search space. Dynamic clause weights were inspired by the method of Lagrange Multipliers (Wu and Wah (2000)) and independently by work on neural networks (Mills and Tsang (2000)). Clause weighting algorithms give state-of-the-art results on many SAT benchmarks. A recent algorithm called Variable Weighting (VW) (Prestwich (2005)) achieves a similar effect by dy-

namically weighting variables instead of clauses. It also introduces a more efficient smoothing technique, and on some problems beats both random walk and clause weighting algorithms.

Some SAT local search algorithms have been adapted to integer programs. WSAT(PB) (Walser (1997)) is a generalisation of the Walksat algorithm that applies to 0/1 problems, and WSAT(OIP) (Walser *et al.* (1998)) is a further generalisation to integer variables. Saturn (Prestwich (2004)) is a hybrid of local search and constraint propagation for 0/1 problems, generalised from an earlier SAT algorithm. The time seems ripe for exploiting the latest SAT developments by adapting a more modern algorithm. We believe that these algorithms are more suitable than, say, simulated annealing for problems in which feasibility is hard to establish, as they have been developed specifically for such problems. We shall adapt VW to an algorithm for full integer linear programs, which we call VWILP.

#### 4.1 *The new algorithm*

Without loss of generality we assume that all constraints are of the form  $\sum_i w_i v_i \leq k$ . Constraints of the form  $\geq$  can be transformed to  $\leq$  by reversing signs, and an = constraint can be decomposed to two inequalities. Similarly we shall assume that all problems are minimisation problems, as maximisation problems can be transformed to minimisation by reversing signs in the objective function.

VWILP is summarised in Figure 1. It begins by assigning randomly-chosen domain values to all variables, and setting an initial upper bound  $U_0 \leq \infty$  on the linear objective function  $f$  to be minimised. It then finds a feasible solution subject to the *cost constraint*  $f < U_0$ . Feasibility is established by flipping values assigned to variables in violated constraints, until no violations remain (the cost constraint is treated like any other constraint here). On establishing feasibility  $U_i$  is reduced to a value  $U_{i+1}$  chosen so that the cost constraint is just violated, then the process restarts with variables initialised to the assigned values from the previous solution. Thus VWILP generates an improving series of feasible solutions, and this process continues until either a time threshold or a known lower bound is reached. We now provide details on the heuristics used by VWILP.

#### 4.2 *Variable selection*

The kernel of VWILP is its inner loop (see Figure 1), which finds a feasible solution to a set of linear constraints on integer variables. The objective function for this subproblem is the number of violated constraints, which must be minimised to zero. The algorithm begins by assigning a randomly-chosen

```

initialise all variables to randomly selected domain values
initialise cost constraint upper bound
repeat until reaching a lower cost bound or time-out
( repeat until no constraint is violated or time-out
  ( randomly select a violated constraint C
    heuristically select a variable V in C
    heuristically flip V to reduce violation degree of C
  )
  reduce upper cost bound to violate cost constraint
)

```

Figure 1. The VWILP algorithm

domain value to each variable. At this point some constraints are violated and others are not. It then repeatedly flips assigned variable values to remove violations. For each flip a violated constraint is randomly selected, and one of its variables is assigned a new value in order to satisfy the constraint — or, if this is not possible, to reduce its *violation degree*. We define the violation degree of a  $\leq$ -constraint as 0 if it is satisfied, otherwise as its left hand side minus its right hand side. For example, suppose that a constraint  $a + b + c \leq 2$  is violated because  $a = b = c = 1$ , where variables  $a, b, c$  are integer variables with domain  $\{0, 1\}$ . This constraint has violation degree 1 and can be satisfied by flipping any variable to 0. But no single flip will satisfy the constraint  $a + b + c \leq 1$  which has violation degree 2: at least two variables must be flipped to 0. However, a single flip to 0 reduces the violation degree of the constraint to 1. (Note that at least one improving flip is possible from any violated constraint; if not then the constraint can never be satisfied.)

From a violated constraint, the choice of which improving flip to make is guided by a heuristic designed to minimise the number of constraints whose violation degree is increased by the flip. For each variable  $v$  we maintain two integers  $I_v$  and  $D_v$ . Integer  $I_v$  is defined as the number of constraints whose violation degree would be increased, minus the number of constraints whose violation degree would be decreased, if the value assigned to  $v$  were to be incremented by 1. Integer  $D_v$  denotes the same information if  $v$  were to be decremented by 1. These integers are maintained during search: on flipping a variable, each constraint in which it occurs is visited, and the integers associated with the other variables in the constraint are adjusted. Now on randomly selecting a violated constraint  $\sum_i w_i v_i \leq k$  the flip heuristic selects the variable in the constraint with least *penalty*  $P_v$ , where the penalty is defined as follows. If  $w_v > 0$  then  $v$  must be flipped to a lower value in order to reduce the constraint's violation degree. If  $v$  already has its lowest possible value then it has penalty  $P_v = \infty$ , otherwise it has penalty  $P_v = D_v$ . Conversely, if  $w_v < 0$  then  $v$  must be flipped to a higher value. If it already has its greatest value then it has penalty  $P_v = \infty$ , otherwise it has penalty  $P_v = I_v$ .

Note that  $P_v$  gives a reasonable estimate of the effect of a flip on already-violated constraints: it indicates how many constraints will become more or less violated if  $v$  is incremented or decremented. Moreover, if  $v$  is to be incremented or decremented by exactly 1 (which is always true of binary variables) then  $P_v$  also gives an accurate picture of the effect of flipping  $v$  on satisfied constraints: if the flip violates a constraint then  $P_v$  detects this. However, if another value is to be assigned to  $v$  then  $P_v$  is less accurate. For example, we may have  $I_v = 0$  meaning that incrementing the value of  $v$  by 1 has no effect on the violation degree of any other constraint. But it may be the case that incrementing it by 2 violates many satisfied constraints. More accurate information could be maintained to handle large changes in value, but this would be more expensive to maintain during search.

### 4.3 Value selection

Having selected a variable  $v_i$  in a violated constraint to flip, we must decide what value to flip it to. Suppose the left hand side of the constraint currently has value  $s$ , the right hand side is  $k$ , the weight of  $v_i$  is  $w_i$ ,  $v_i$  currently has value  $a_i$ , and the domain of  $v_i$  is the closed interval  $[L_i, U_i]$ . Then flipping  $v$  to value  $a'_i = a_i - (k - s)/w_i$  would just satisfy the constraint. However, if  $a'_i$  is outside the domain of  $v_i$  then we reduce the violation degree of the constraint by choosing  $L_v$  if  $a'_i < L_i$  and  $U_v$  if  $a'_i > U_i$ . The aim of this value selection heuristic is to choose the value for the chosen variable that just satisfies, or most nearly satisfies, the chosen violated constraint.

As far as we know, this is a new value selection heuristic for local search on integer models. The WSAT(OIP) algorithm instead makes small changes to the value of  $v_i$  to avoid large perturbations of the current state. In experiments our heuristic gave better results than a WSAT(OIP)-like heuristic and other variations.

### 4.4 A diversification heuristic

A further technique from the VW algorithm is added to VWILP, to improve its ability to escape from local minima. Instead of selecting the variable with smallest  $P_v$  it selects one with smallest value of  $P_v + c(F_v - F/V)$  where  $c$  is a parameter that must be tuned by trial and error (set to 4 in all our experiments),  $F_v$  is the number of times  $v$  has been flipped in the search so far,  $F$  is the total number of flips so far, and  $V$  is the number of variables. The term added to  $P_v$  increases the penalty of variables that have been flipped fewer times than the mean number  $F/V$ , and decreases the penalty of variables that have been flipped more than average. This is analogous to a Tabu heuristic but escapes from some local minima more effectively (Prestwich (2005)).

Table 1. Results with two templates

No	N <sup>a</sup>	S <sup>b</sup>	Scen <sup>c</sup>	Benders			VWILP			$\Delta\%$ cost <sup>f</sup>
				cost	opt <sup>d</sup>	time <sup>e</sup>	cost	opt	time	
1	3	6	10	285.00	y	10	285.00	y	230	0.00
2	3	9	10	285.00	y	8	285.00	y	990	0.00
3	4	2	10	480.00	y	62	480.00	y	4	0.00
4	4	3	10	332.50	y	41	332.50	y	6	0.00
5	4	4	10	322.50	y	190	322.50	y	196	0.00
6	4	5	10	401.00	n	3400	365.00	y	140	9.86
7	4	6	10	308.00	y	670	315.00	n	570	-2.27
8	4	7	10	310.00	y	820	312.50	n	170	-0.80
9	4	8	10	308.00	y	1800	310.00	n	1500	-0.65
10	4	9	10	326.00	n	2900	310.00	n	1680	5.16
11	4	6	11	333.00	y	480	339.00	n	3500	-1.77
12	4	6	12	354.00	y	1100	362.00	n	47	-2.21
13	4	6	13	374.00	y	190	379.50	n	2	-1.45
14	4	6	14	393.50	y	1300	396.00	n	420	-0.63
15	4	6	15	407.00	y	800	414.50	n	3000	-1.81
16	4	6	16	432.50	n	3600	433.25	n	44	-0.17
17	4	6	17	398.25	y	540	400.75	n	220	-0.62
18	4	6	18	377.75	y	290	380.75	n	740	-0.79
19	4	6	19	396.75	y	1200	414.25	n	1700	-4.22
20	4	6	20	406.75	y	400	409.25	n	110	-0.61

<sup>a</sup>the number of variations

<sup>b</sup>the number of slots

<sup>c</sup>the number of demand scenarios

<sup>d</sup> *y*: optimality proved, *n*: not optimal, *?*: optimal solution is unknown

<sup>e</sup> solution time is given in seconds (max. allowed solution time is 1 hour),

<sup>f</sup>  $\Delta = 100(VWILP - Benders)/VWILP$

## 5 Experimental results

We use the twenty benchmarks from Tarim and Miguel (2005), each having two templates. We compare our algorithm with the best Benders' Decomposition approach used in Tarim and Miguel (2005) (with single cuts), which was easily the most efficient compared to two other Benders variants (complete and multiple cuts) and a Stochastic Constraint Programming method. For each benchmark we allow Benders and VWILP to run for a maximum of one hour on a 2GHz Intel Centrino, 1GB RAM machine. Benders is a complete method so it may terminate earlier if it finds an optimal solution. VWILP is incomplete but we terminate it earlier if it reaches a known optimal solution (found by Benders). VWILP used an initial upper bound for each instance, calculated by adopting a *zero-production policy*. In our zero-production policy the number of prints for each template is set to zero; in other words, it is planned to have a backlog (or expedite) and incur a shortage cost for any realised demand. It is clear that, although the zero-production policy is an inferior one, it always yields a valid upper bound for any instance.

As explained in Section 4, we adapt VW to an algorithm called VWILP for full integer linear programs, in which decision variables are required to have finite domains. In order to comply with this assumption all continuous

variables are declared as integers via a scaling-up of 100, hence guaranteeing an accuracy of  $\frac{1}{100}$  of a unit, and restricted to values 0 – 9,999.

The results in Table 1 show that VWILP is not very competitive with the best Benders method at finding optimum solutions. The  $\Delta$  figures show the percentage improvement of the VWILP solution over the Benders solution, or vice-versa if the figure is negative. Only in two of these results does VWILP beat Benders ( $\Delta > 0$ ); on the five smallest problems neither algorithm dominates the other. However, a different picture emerges when we compare times taken to find near-optimal solutions. The graphs in Figure 2 compare the convergence of the two algorithms on three instances: 1, 10 and 20. In each case VWILP finds better solutions than Benders for most of the time: that is, if the algorithms are halted at a randomly-chosen time, then VWILP is likely to return a better solution than Benders. VWILP is therefore superior as an *anytime algorithm*, that is an algorithm that returns the best answer possible even if it is not allowed to run to completion, and may improve on the answer if it is allowed to run longer. Anytime algorithms are a pragmatic approach to time-critical optimisation problems.

Table 2. Results with three templates

No	N	S	Scen	Benders			VWILP			$\Delta\%$ cost
				cost	opt	time	cost	opt	time	
1	3	6	10	295.50	n	68	285.00	?	160	3.68
2	3	9	10	305.50	n	110	285.00	?	440	7.19
3	4	2	10	309.00	n	3600	307.50	?	2100	0.49
4	4	3	10	462.00	n	110	310.00	?	99	49.03
5	4	4	10	465.00	n	440	310.00	?	690	50.00
6	4	5	10	481.00	n	180	365.00	?	310	31.78
7	4	6	10	805.00	n	1100	307.50	?	1600	161.79
8	4	7	10	464.50	n	1100	312.50	n	230	48.64
9	4	8	10	471.00	n	1600	312.50	n	390	50.72
10	4	9	10	775.00	n	1500	310.00	?	3100	150.00
11	4	6	11	770.00	n	620	333.00	?	87	131.23
12	4	6	12	747.25	n	630	361.75	n	3300	106.57
13	4	6	13	557.00	n	290	372.00	?	2200	49.73
14	4	6	14	522.25	n	420	393.50	?	1300	32.72
15	4	6	15	531.25	n	410	407.00	?	570	30.53
16	4	6	16	544.25	n	820	428.25	?	2900	27.09
17	4	6	17	526.50	n	300	399.50	n	2200	31.79
18	4	6	18	512.75	n	220	375.75	?	3000	36.46
19	4	6	19	530.75	n	620	399.25	n	880	32.94
20	4	6	20	547.25	n	660	408.00	n	2900	34.13

Next we increase the number of templates to three and repeat the experiments, with results shown in Table 2. Benders finds these problems much more difficult, and after an hour the quality of its solutions is poorer than before. In contrast, VWILP finds *better* solutions than before. (We do not know the optimum solutions to these problems, but with more available templates the optimum solution quality is obviously at least as good.) Finally, we increase the number of templates to four and repeat the experiment again, with results

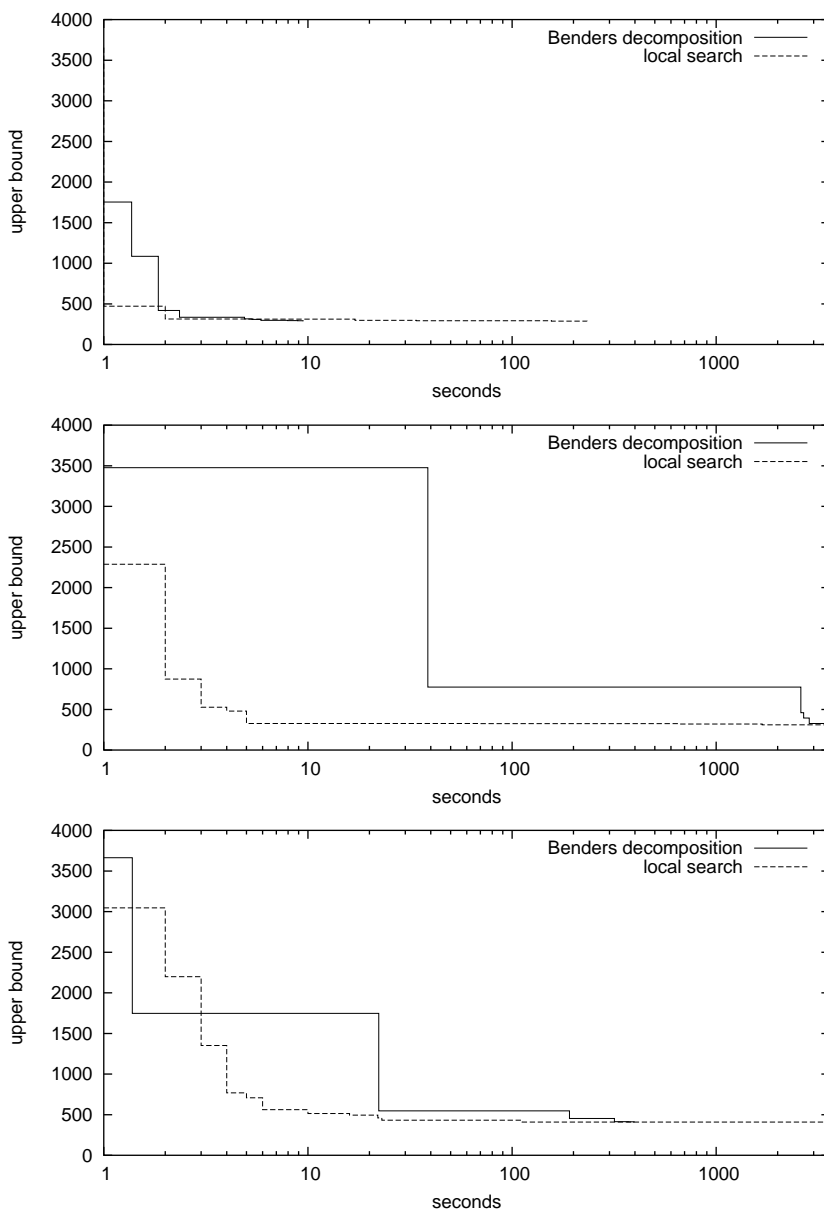


Figure 2. Convergence on instances 1, 10 and 20

shown in Table 3. Benders now performs even worse than with three templates, while VWILP performs almost identically. These results clearly demonstrate the superior scalability of VWILP.

There is another difference between Benders and VWILP that is not ap-

Table 3. Results with four templates

No	N	S	Scen	Benders			VWILP			$\Delta\%$ cost
				cost	opt	time <sup>a</sup>	cost	opt	time	
1	3	6	10	295.50	n	860	285.00	?	470	3.68
2	3	9	10	545.00	n	82	286.00	n	3100	90.56
3	4	2	10	2128.00	n	2100	310.00	n	35	586.45
4	4	3	10	468.00	n	1100	307.50	?	2300	52.20
5	4	4	10	3478.00	n	0.0	310.00	?	210	1021.94
6	4	5	10	481.00	n	93	365.00	?	430	31.78
7	4	6	10	855.50	n	650	307.50	?	950	178.21
8	4	7	10	2563.00	n	140	312.50	n	1700	720.16
9	4	8	10	3478.00	n	0.0	312.50	n	320	1012.96
10	4	9	10	3476.00	n	0.0	310.00	?	2200	1021.29
11	4	6	11	770.00	n	3300	335.00	n	1700	129.85
12	4	6	12	1569.50	n	380	357.00	n	1900	339.64
13	4	6	13	557.00	n	1500	372.00	?	1200	49.73
14	4	6	14	522.25	n	2200	393.50	?	1100	32.72
15	4	6	15	531.25	n	2200	407.00	?	2700	30.53
16	4	6	16	2003.75	n	220	433.25	n	2400	362.49
17	4	6	17	526.50	n	1500	397.00	?	3200	32.62
18	4	6	18	512.75	n	1100	379.50	n	1500	35.11
19	4	6	19	530.75	n	3600	399.25	n	780	32.94
20	4	6	20	1747.50	n	170	411.75	n	650	324.41

<sup>a</sup> In three Benders-instances, {5,9,10}, the initial feasible solutions, which are practically found at  $t = 0$ , could not be improved at all in the allowed 1-hour solution time

parent in our tables and graphs. Benders tends to make large improvements in quite regular steps, whereas VWILP makes many small improvements in rapid bursts. For problems in which the initial upper bound is very high this is a drawback with VWILP, though our use of a zero-production policy upper bound helps significantly. SAT algorithms have been developed for problems in which feasibility is hard to establish, but more optimisation-oriented heuristics might accelerate convergence.

## 6 Conclusion

We presented a new ILP model for a problem in production planning under uncertain demand, designed specifically for use by local search algorithms. We then described a local search algorithm for ILP called VWILP, based on a state-of-the-art SAT algorithm, and applied it to our model. On smaller problem instances it was beaten by the best known Benders' Decomposition method when used to find known optimal solutions, but was superior when used as an anytime algorithm to find near-optimal solutions in a limited time. Moreover, on larger instances it scaled far better than Benders, and is the only practicable approach we know of for finding high-quality solutions to such problems. Because VWILP is a general-purpose algorithm it can also be applied to other problems, as long as they can be modelled as ILP. We

therefore propose it, and similar future algorithms, as general-purpose tools for production planning and related problems.

Modelling is more of an art than a science, and a good model for one algorithm may be very poor for another. The ILP we used was poor from both the branch-and-bound and constraint programming points of view, because it is a highly-symmetric big-M formulation. However, it turns out to be very well-suited for local search. There are interesting modelling issues here, and a detailed study of alternative models for local search seems overdue (in the spirit of Ormond and Williams (2004) for example). Little research has been done on what makes a good model for local search, but for first attempts along these lines see Prestwich (2003), Prestwich and Roli (2005).

In future work we intend to generalise VWILP to handle continuous variables and non-linear constraints, so that more compact and natural problem models can be used. Non-linear constraints often arise naturally when transforming probabilistic problems to deterministic ones, so this will enable us to experiment with more interesting production problems.

### *Acknowledgements*

S. Armagan Tarim is partially supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR). This material is also based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075.

### REFERENCES

- ABRAMSON, D., DANG, H. and KRISHNAMOORTHY, M., A Comparison of Two Methods for Solving 0–1 Integer Programs Using a General Purpose Simulated Annealing Algorithm. *Annals of Operations Research* vol. 63, 1996, pp. 129–150.
- ABRAMSON, D. and RANDALL, M., A Simulated Annealing Code for General Integer Linear Programs. *Annals of Operations Research* vol. 86, 1999, pp. 3–24.
- CLARK, A.R., A Local Search Approach to Lot Sequencing and Sizing. *Third International Workshop of the IFIP*, Firenze, 2002, pp. 145–152.
- CONNOLLY, D., General Purpose Simulated Annealing. *Journal of the Operational Research Society* vol. 43, 1992, pp. 495–505.
- HOOS, H.H. and STUTZLE, T., *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

- LIN, S. and KERNIGHAN, B.W., An Effective Heuristic for the Traveling Salesman Problem. *Operations Research* vol. 21, 1973, pp. 498–516.
- LOKKETANGEN, A. and GLOVER, F., Tabu Search for Zero-One Mixed Integer Programming with Advanced Level Strategies and Learning. *International Journal of Operations and Quantitative Management* vol. 1, no. 2, 1995, pp. 89–108.
- MCALLESTER, D.A., SELMAN, B. and KAUTZ, H.A., Evidence for Invariants in Local Search. *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / MIT Press 1997, pp. 321–326.
- MILLS, P. and TSANG, E.P.K., Guided Local Search for Solving SAT and Weighted MAX-SAT Problems. *Journal of Automated Reasoning* vol. 24, Kluwer, 2000, pp. 205–223.
- NETO, T. and PEDROSO, J.P., GRASP for Linear Integer Programming. *Metaheuristics: Computer Decision-Making*. Kluwer Academic Publishers, 2004, pp. 545–574.
- ORMOND, A.J. and WILLIAMS, H.P., A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem. Working Paper No. LSEOR 04.67, London School of Economics and Political Science, UK, 2004.
- PEDROSO, J.P., An Evolutionary Solver for Linear Integer Programming. BSIS Technical Report 98-7, Riken Brain Science Institute, Wako-shi, Saitama, Japan, 1998.
- PRESTWICH, S.D., Random Walk With Continuously Smoothed Variable Weights. *Eighth International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3569, Springer, 2005, pp. 203–215.
- PRESTWICH, S.D., Incomplete Dynamic Backtracking for Linear Pseudo-Boolean Problems. *Annals of Operations Research* vol. 130, 2004, pp. 57–73.
- PRESTWICH, S.D., Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research* vol. 118, 2003, pp. 137–150.
- PRESTWICH, S.D. and ROLI, A., Symmetry Breaking and Local Search Spaces. *Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer, 2005, pp. 273–287.
- PROLL, L. and SMITH, B.M., Integer Linear Programming and Constraint Programming Approaches to a Template Design Problem. *INFORMS Journal of Computing* vol. 10, 1998, pp. 265–275.
- SELMAN, B., KAUTZ, H.A. and COHEN, B., Noise Strategies for Improving Local Search. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press, 1994, pp. 337–343.

- TARIM, S.A. and MIGUEL, I., A Hybrid Benders' Decomposition Method for Solving Stochastic Constraint Programs with Linear Recourse. *Joint Annual Workshop of ERCIM/CoLogNet*, 2005.
- WALSER, J.P., Solving Linear Pseudo-Boolean Constraint Problems with Local Search. *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, AAAI Press / MIT Press, 1997, pp. 269–274.
- WALSER, J.P., IYER, R. and VENKATASUBRAMANYAN, N., An Integer Local Search Method with Application to Capacitated Production Planning. *Fifteenth National Conference on Artificial Intelligence*, AAAI Press, 1998, pp. 373–379.
- WU, Z. and WAH, B.W., An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, AAAI Press, 2000, pp. 310–315.