A co-operative parallel heuristic for mixed zero-one linear programming: Combining simulated annealing with branch & bound

V. Nwana, K. Darby-Dowman and G. Mitra
Department of Mathematical Sciences
Brunel University
West London
UB8 3PH
United Kingdom

June 15, 2003

Abstract

This paper considers the exact approach of branch and bound (B&B) and the metaheuristic known as simulated annealing (SA) for processing integer programs (IP). We extend an existing SA implementation (GPSIMAN) for pure zero-one integer programs (PZIP) to process a wider class of IP models, namely mixed zero-one integer programs (MZIP). The extensions are based on depth-first B&B searches at different points within the SA framework. We refer to the resultant SA implementation as MIPSA. Furthermore, we have exploited the use of parallel computers by designing a co-operative parallel heuristic whereby concurrent executions of B&B and MIPSA, linked through a parallel computer, exchange information in order to influence their searches. Results reported for a wide range of models taken from a library of MIP benchmarks demonstrate the effectiveness of using a parallel computing approach which combines B&B with SA.

Keywords: Branch and Bound, Integer Programming, Metaheuristics, Parallel Computing, Simulated Annealing

1 Introduction

Branch and bound (B&B) remains the method of choice for processing Integer Programming (IP) and Mixed Integer Programming (MIP) models [10]. Although there have been major algorithmic advances within B&B as well as considerable improvements in computing power, the overall usefulness of B&B for processing IP models is sometimes limited by unsatisfactory solution times. In such cases, B&B is often used in a heuristic way whereby good solutions are accepted in place of proven optimal ones.

General purpose heuristics or metaheuristics such as Simulated Annealing (SA) are increasingly being applied to process IP and MIP models [4],[20]. Optimality is not a design goal of such approaches. Instead, they aim to find good solutions quickly and at a reasonable computing cost.

A major difference between the B&B and SA approaches lies in the criteria that control the sequence of feasible solutions obtained. B&B employs a monotonic search process such that only solutions that improve on the incumbent one are accepted. SA, on the other hand, can consider solutions that are inferior to the incumbent. This is an attempt to avoid convergence to a local optimum. With both B&B and SA, very little is known about the structure of the search space at the start of an execution run and information gained during the solution process is used to give informed direction to the search. In parallel executions of both approaches this information is shared between the processors that investigate different regions of the search space. There are several implementations of parallel B&B (PB&B) algorithms to solve IP problems [2] [12] [14] [17]. An example of a parallel SA implementation for solving IP problems is given in Lee and Stiles[13].

The main focus of this paper is the investigation of a hybrid approach which shares information between concurrent executions of B&B and SA. Hybrid approaches for solving optimisation problems have been proposed in recent years. For example, Darby-Dowman and Little [5] identify three hybridization structures for Constraint Logic Programming (CLP) and B&B, using:

(i) a two stage process whereby one approach is applied and, at some point, the partial results are passed to the other approach to complete the solution process,

(ii) a CLP search whereby a good integer feasible solution is obtained, which provides a good bound and possibly a good branching agenda for B&B tree development, and

(iii) a CLP master system which generates sub-problems to be solved by B&B.

There are several examples of hybrid algorithms that conform to one of these three structures. Reeves and Hohn [21] design a genetic algorithm (GA) framework which can incorporate any local neighbourhood search technique. This approach conforms to hybrid structure (i) above. French *et al.* [9] describe a hybrid genetic–B&B for the Travelling Salesman Problem (TSP) which conforms to both hybrid structures (i) and (ii) above. Moscato [18] reports a successful hybridisation of properties of Tabu Search with those of global optimisation. This is a case of hybrid strategy (ii) above. Teghem *et al.* [22] also uses the second hybridisation strategy by embedding a linear programming solver to deal with continuous variables within a simulated annealing framework.

In the hybrid B&B and SA approach that we present in this paper, we use the last two hybridization structures suggested by Darby-Dowman and Little [5]. We begin in section 2 by defining the general MIP and highlight special cases. We proceed in section 3 to investigate the components of an implementation of SA, (GPSIMAN) [4], for pure zero-one IPs. In section 4, we discuss the first level of hybridization, which involves incorporating depth-first B&B search at different points in an SA execution run. We call the resultant procedure MIPSA. MIPSA extends GPSIMAN in that it can be applied to a wider class of integer programs, namely, mixed zero-one integer programs, MZIP. We proceed to describe a co-operative (parallel) framework that exploits the use of information obtained by concurrent executions of SA and B&B. We call this procedure PACO. Results are reported for a set of computational experiments designed to assess the effectiveness of MIPSA and PACO. Finally in section 5, we summarize our conclusions.

2 Mixed Integer Programming: Problem Definition

The Mixed Integer Program (MIP) is defined using the following *index sets* and *model coefficients*.

Index sets

$B = \{1, \cdots, \|B\|\}$	Index set for binary variables
$I = \{\|B\|+1, \cdots, \|B\|+\|I\|\}$	Index set for integer variables
$C = \{\|B\|+\|I\|+1, \cdots, \|B\|+\|I\|+\|C\|\}$	Index set for continuous variables
$N = B \cup I \cup C$	Index set for all variables
$M = 1, \cdots, \|M\|$	Index set for all constraints

Model coefficients

$l_j, u_j, c_j, a_{ij}, b_i, (i \in M, j \in N)$ are known constants.

The general MIP is defined as

$$Min \qquad Z_{MIP} = \sum_{j \in B} c_j x_j + \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \tag{1}$$

subject to

$$\sum_{j \in B} a_{ij} x_j + \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, |M| \tag{2}$$

and

$$x_j \geq 0 \qquad j \in N$$

If $B, I = \oslash$ then $N = C$. This problem is a *linear program* (LP).

If $I, C = \oslash$ then $N = B$. This problem is called a Pure Zero-One Integer Problem (PZIP).

If $C = \oslash$ then $N = B \cup I$. This problem defines a Pure Integer Program (PIP).

If $I = \oslash$ then $N = B \cup C$. This problem is referred to as Mixed Zero-One Integer Program (MZIP). PZIP and MZIP models constitute the majority of IP models in practice. In fact, profiling the library of MIP benchmarks, MIPLIB [1] shows only one IP model that does not belong to one of these two classes.

In this paper, we concentrate on solution approaches for processing MZIP models. Two such approaches that are widely used in practice are simulated annealing (SA) and branch and bound (B&B). It is assumed that the reader is familiar with both these approaches. References for SA applied to integer programs include: Collins *et al.* [3], Dowsland [6], Eglese [7], Kirkpatrick *et al.* [11] and Lundy and Mees [15]. Reference material for B&B abound, including: Johnson *et al.* [10], Linderoth and Savelsbergh [14] and Mitra [16].

3 Connolly's SA Heuristic for PZIPs

Connolly's SA code (GPSIMAN) [4] accepts a model in the conventional MPS format for PZIP. It generates an initial feasible solution using a pseudo-random approach and proceeds to explore the search space by flipping variables (changing their values either from 0 to 1 or from 1 to 0) randomly and measuring the change in the objective function (cost) value. A special feature of the heuristic is that it implements a reheating schedule in which a number of annealing runs are performed. The first of these runs calculates an appropriate initial and final temperature for the next run. The temperature T follows a scheme proposed by Lundy and Mees [15] whereby the recurrence relation $T_{n+1} := \frac{T_n}{1+\alpha T_n}$ is used to obtain the temperature at which annealing run $n+1$ is performed based on the temperature T_n of annealing run n. The value of α is $\frac{T_{initial} - T_{final}}{M.T_{initial}.T_{final}}$, where $T_{initial}$, T_{final}, and M represent the initial temperature, final temperature and the maximum number of iterations and are specified by the user. α therefore ensures that the temperature falls from $T_{initial}$ to T_{final} in exactly M iterations.

Connolly's heuristic for an n-variable PZIP is presented in *Algorithm 1*. The heuristic operates as follows. The model and SA parameters are initialised; an initial solution is generated and the cost computed. Several (user-specified) annealing runs are performed. During each run, variables are chosen and assigned new values at random. Since the heuristic was designed for PZIPs, assignment of a new value to a variable is tantamount to flipping a variable to its lower or upper bound (0 or 1). The effect on the change in the objective cost of flipping the randomly chosen variable is then measured. However, since these changes do not guard against infeasibility, feasibility may be lost by flipping a variable. The heuristic has a feasibility restoration technique (RESTORE) whereby variables other than the most-recently flipped variable (that caused infeasibility) are flipped in order to obtain a new feasible solution. The steps of the RESTORE technique are presented in Algorithm 2. Embedded in RESTORE is a variable calibration technique called GETSWOP in which a 'help-score' is computed for each variable based on how helpful a change in its value would be for restoring feasibility. The rationale of GETSWOP is to identify the variable which not only helps to restore the feasibility of the row currently under consideration but also establishes the 'criticality' of that row within the context of the feasibility of the whole problem. Further details of the GETSWOP procedure are given in Connolly [4].

The most helpful variable is flipped and if feasibility is not restored, the second most helpful variable is considered. This procedure continues in a depth first manner with some backtracking capabilities. If the procedure cannot restore feasibility after a pre-defined fixed number of flips, then the original move is rejected. Otherwise, the change in the cost is calculated. If the current solution is better than the incumbent solution then it is accepted as the new imcumbent solution if the SA acceptance function is satisfied (i.e. if $e^{-\Delta c/T} < R$) and rejected otherwise. If it is rejected and a number of consecutive previous solutions have also been rejected, then annealing is performed at the temperature at which the best solution in this trial was found. When the annealing run has ended, the temperature is reheated following a cooling schedule to a level slightly lower than in the previous run. This process continues for a user-specified number of annealing trials.

Algorithm 1 SA Algorithm of Connolly For an N-Variable PZIP Minimisation Problem[4]

```
Step 1: Initialisation
Get the SA parameters; Initial Temperature = $T_{initial}$, Final Temperature = $T_{final}$,
Number of iterations per trial = $M$, Maximum number of rejected moves = $MR$
Read problem in MPS format
Generate initial feasible solution $\mathbf{x}$
Set $\mathbf{x^{initial}} := \mathbf{x}$
Compute cost of initial solution $c(\mathbf{x^{initial}})$
Set Temperature $T := T_{initial}$, Iteration counter $K := 0$ and Temperature degradation
constant $\alpha := \frac{T_{initial}-T_{final}}{MT_{initial}T_{final}}$

Step 2: Annealing Trials
while $K < M$ do
  while $T > T_{final}$ do
    $K := K + 1$
    $i = K \quad MOD \quad N$
    $\mathbf{x}^{'} = \mathbf{x}$
    $\mathbf{x}_i^{'} = 1 - \mathbf{x}_i$
    Restore feasibility($\mathbf{x}_i^{'}$)
    Calculate $\Delta c = c(\mathbf{x}^{'}) - c(\mathbf{x})$
    $R = unif\_rand(0, 1)$
    if $\Delta c > 0$ and $e^{-\Delta c/T} < R$ then
      $Rejected\_moves = Rejected\_moves + 1$
      if $Rejected\_moves > MR$ then
        $\alpha = 0$ and $T = T_{best}$
      end if
    else {Accept the move}
      $\mathbf{x} = \mathbf{x}^{'}$; $Rejected\_moves = 0$
    end if
    if $c(\mathbf{x}) < c_{best}$ then
      $c_{best} = c(\mathbf{x})$ and $T_{best} = T$
    end if
  $T = \frac{T}{1+\alpha T}$
  end while
end while
```

Algorithm 2 RESTORE routine

```
*Step 1: Initialisation*

Set the initial parameters: Most recently flipped variable $\mathbf{x_{orig}}$, Depth level = *level*, Feasibility obtained = $feas_{ok}$, Maximum number of iterations to suggest infeasibity after move = $MFI$, Current solution = $\mathbf{x^{current}}$
Set level = 1: $feas_{ok}$ = false; $\mathbf{x_{orig}} = \mathbf{x_i}$; $MFI = M$, $M \in \mathbf{Z}$; $\mathbf{x^{current}} = \mathbf{x}$
*Step 2: Feasibility check*
if $x^{current}$ is feasible then
  *Step 2.1: Return - the current solution is feasible*

else
  *Step 2.2 The current solution is not feasible*
  Call the GETSWOP routine to choose the most helpful column, $\mathbf{x_{good}}$

  if $x_{good} = \{\}$ then
    *Step 2.2.1 No column to swap*
    $feas_{ok}$ = false
    Return
  else
    *Step 2:2.2 Flip a column from* 0 *to* 1 *or vice versa*
    Set the column to be flipped: $\mathbf{x_{flip}}$=$\mathbf{x_{good}}$
    Update $\mathbf{x^{current}}$ with the updated value of $\mathbf{x_{flip}}$
    Level = Level+1
    $Iteration\quad count = 0$
    while $Iteration\quad count < M$ do
      $Iteration\quad count = Iteration\quad count + 1$
      if $\mathbf{x^{current}}$ is feasible then
        Go to *Step 2.1*
      else
        Go to *Step 2.2*
      end if
    end while
  end if
  if $MFI \geq M$ then
    *Step 2.2.2.1 Feasibility not established*
    if Level = 1 then
      Return - Feasibility cannot be restored
    else
      Backtrack
      Level = Level -1
      Block the variable flip taken in *Step 2.2.2* at this level
      GOTO STEP 2
    end if
  end if
end if
```

4 Extending Simulated Annealing to Process MZIP – MIPSA

4.1 Motivation

Connolly [4] applied GPSIMAN on a limited number of PZIP models. One of the aims of MIPSA is to process a wider range of models than PZIPs. We have therefore extended the GPSIMAN algorithm to solve MZIP models. These extensions are based on the following observations of the original GPSIMAN algorithm.

(i) The mechanism for generating an initial integer feasible solution to the PZIP is rudimentary. The initial solution is obtained by randomly enumerating the variables to their 0-1 states and testing for feasibility. The problem of obtaining an initial feasible solution for a MZIP problem is more challenging since continuous variables are present.

(ii) GPSIMAN can be viewed as a modular framework in which each procedure can be independently amended or replaced without destroying the overall logic of the algorithm.

(iii) GPSIMAN employs a feasibility restoration routine (*RESTORE*) during its neighbourhood transition process. Since SA is effectively a series of local searches based on new starting solutions, we can by-pass feasibility restoration by providing the search with an alternative feasible starting solution obtained from some other process such as B&B.

4.2 Statement of MIPSA heuristic

Our implementation of SA for MZIP, MIPSA, involves a hybrid of local search and a depth-first B&B approach. However, B&B is only used as a feasible solution generator. The SA framework is maintained and B&B is called at the beginning of the algorithm, as well as periodically during its execution to provide integer solutions about which SA performs local search. The steps of the resultant algorithm are described in *algorithm 3* below.

Algorithm 3 Statement of the MIPSA heuristic

```
Step 1: Initialisation
Get the SA parameters; Initial Temperature = $T_{initial}$, Final Temperature = $T_{final}$,
Number of iterations per trial $M$, Maximum number of rejected moves = $MR$
Read problem in MPS format
Set $\mathbf{x}^{\mathbf{initial}} = \mathbf{x}$
Compute cost of initial solution $c(\mathbf{x}^{\mathbf{initial}})$
Set Temperature $T := T_{initial}$, Iteration counter $K := 0$ and Temperature degradation
constant $\alpha := \frac{T_{initial} - T_{final}}{M T_{initial} T_{final}}$
Generate initial feasible solution x by calling a directed depth first B&B
search

Step 2: Annealing Trials
while $K < M$ do
  while $T > T_{final}$ do
    $K := K + 1$
    $i = K \quad MOD \quad N$
    $\mathbf{x}^{'} = \mathbf{x}$
    $\mathbf{x}_i^{'} = 1 - \mathbf{x}_i$
    Check Feasibility
    if x is infeasible then
      Call a directed depth-first B&B to generate a new feasible neighbour-
      hood solution. The B&B tree is then abandoned,
      Calculate $\Delta c = c(\mathbf{x}^{'}) - c(\mathbf{x})$
      $R = unif\_rand(0, 1)$
      if $\Delta c > 0$ and $e^{-\Delta c/T} < R$ then
        $Rejected\_moves = Rejected\_moves + 1$
        if $Rejected\_moves > MR$ then
          $b = 0$ and $T = T_{best}$
        end if
      else {Accept the move}
        $\mathbf{x} = \mathbf{x}^{'}$; $Rejected\_moves = 0$
      end if
      if $c(\mathbf{x}) < c_{best}$ then
        $c_{best} = c(\mathbf{x})$ and $T_{best} = T$
      end if
    $T = \frac{T}{1 + \alpha T}$
    end if
  end while
end while
```

In summary, MIPSA maintains the GPSIMAN framework and replaces the initialisation and feasibility restoration techniques in GPSIMAN by employing independent B&B tree searches. The MIPSA heuristic extends GPSIMAN to process MZIP problems since, in general, B&B solvers can deal with more generic models than the PZIPs considered by GPSIMAN. It is worth noting that MIPSA deals with continuous variables only within the B&B implementation: the binary variables in the MZIP are fixed and the corresponding values for the continuous variables are computed by calling a linear programming routine. This approach is effective in practice since the ratio of binary variables to continuous variables is generally very high for most MZIP models in practice. A similar approach was used by Teghem *et al.* [22].

4.3 A Parallel Co-operative MIPSA and B&B Approach

It is well known that MIP models are often difficult to solve. Parallel computing provides a platform that may be used to improve the solution times of difficult models. There are several implementations of parallel B&B algorithms [2] [12] [14] [17]. An example of a parallel SA code for processing MIP is given in [13]. Most of these implementations either partition the search space of the problem for investigation by different processors (data parallelism) or distribute different aspects of the approaches to be performed by different processors (functional parallelism).

There has been little focus, however, on employing alternative solution methods which can exploit the parallel framework by exchanging information to guide their search procedures. We have implemented a parallel heuristic, referred to as PACO, which involves concurrent executions of B&B and MIPSA, on different computers connected in a parallel framework using *Parallel Virtual Machine* (PVM). The two solution procedures exchange search-related information, namely, *bounds* and *variable choice information.*

Upper bounds are exchanged between the B&B code and the SA code in the following way. Upper bounds obtained from SA are immediately passed onto the B&B code. If these bounds are better than B&B's incumbent bound, the B&B execution updates its "current best bound" value. Moreover, any integer bound obtained by the B&B execution is passed on to the SA code and used as an alternative "reheated solution". The bounds of B&B are equivalent to the cost in the SA code. If the objective function value of the new integer solution obtained by B&B is better than the incumbent

SA objective cost, then the improved integer solution is accepted. SA then effectively performs a local search around the new feasible solution. It then passes on improved feasible solutions in the neighbourhood of the existing feasible solution to the B&B code.

Secondly, variable choice information is exchanged between the two codes as follows. MIPSA uses GPSIMAN's variable calibration technique whereby the different 0-1 variables are ranked according to their usefulness in restoring feasibility. This bears similarities to the notion of pseudocosts when implemented within B&B, since pseudocosts attempt to capture the most helpful variable for branching. Once both SA and B&B have calculated their respective variable ranking and pseudocosts information, a single variable choice list is constructed by averaging the ranking positions from both strategies. This strategy represents the overall *variable choice heuristic* for the parallel heuristic.

4.4 Experimental results and analyses

Computational experiments were carried out to investigate the performances of GPSIMAN, MIPSA, B&B and the parallel co-operative approach, PACO, for both PZIP and MZIP models. When run in serial mode, the machine architecture was a Pentium III 500MHz computer with 128MB of RAM. The parallel co-operative heuristic was run on two Pentium III 500MHz computers connected through a 100Mbit Ethernet network. The parallel architecture is therefore a distributed memory model linked together through the *parallel virtual machine* (PVM) message passing system. PVM is software that permits a network of heterogeneous computers to be used as a single large parallel computer. Our co-operative heuristic exploits this architecture as follows. MIPSA is run on one machine and independently generates search information (bounds and variable choice information) to guide its search. Similarly, B&B is executed on the second machine in the parallel architecture. PVM ensures that specific search information can be exchanged between the two computers in order to influence their searches. A summary of the results is presented next.

4.4.1 Summary of Results for PZIP Models

In order to study the performance of MIPSA and PACO, two sets of experiments were conducted on some PZIP models taken from the library of MIP benchmarks, MIPLIB [1]. Summary statistics for the PZIP models considered are shown in *Table 1* below.

Model name	No. of constraints	No. of (binary) variables
air03	124	10757
air04	823	8904
air05	426	7195
cap6000	2176	6000
l152lav	97	1989
lseu	28	89
mitre	2054	10724
mod008	6	315
mod010	146	2655
nw04	36	87482
p0033	16	33
p0201	133	201
p0282	241	282
p0548	176	548
p2756	755	2756
stein27	118	27
stein45	331	45

Table 1: Model statistics of MIPLIB PZIP models

The results presented in *Table 2* represent the two sets of experiments designed to compare the performances of GPSIMAN and MIPSA on PZIP models. The column *IPOPT* holds the optimal (or best available solution) as reported in MIPLIB [1]. *IPBEST* represents the value of the best integer solution found by each execution run. *TT1* and *TTB* represent the times to first and best integer solutions, respectively. In performing the experiments, both GPSIMAN and MIPSA were provided with the same initialisation parameters.

		GPSIMAN			**MIPSA**		
Name	**IPOPT**	**IPBEST**	**TT1**	**TTB**	**IPBEST**	**TT1**	**TTB**
air03	340160	340160	300.01	1489.48	340160	187.4	900.85
air04	56137	56137	476.67	3589.63	56137	400.4	3452.89
air05	26374	26533	412.90	2534.77	26533	388.2	1000.62
cap6000	-2451377	None	N/A	N/A	-2446110	78.34	1542.26
l152lav	4722	4935	411.09	2792.18	4935	188.2	1000.67
lseu	1120	1145	13.80	1865.8	1145	67.30	1123.34
mitre	115155	None	N/A	N/A	124065	360.17	874.3
mod008	307	307	13.4	50.12	307	3.4	43.7
mod010	6548	6665	954.34	954.34	6663	400.3	4872.3
nw04	16862	None	N/A	N/A	16922	63.5	987.4
p0033	3089	3089	46.88	46.88	3089	6.0	10.0
p0201	7615	7615	55.0	104.61	7615	22.0	22.0
p0282	258411	None	N/A	N/A	317275	342.0	1943.0
p0548	8691	9886	76.7	4612.8	9886	87.3	4332.6
p2756	3124	None	N/A	N/A	4083	343.7	5456.6
stein27	28	28	38.1	54.50	28	8.6	16.7
stein45	30	30	21.89	46.72	30	10.2	29.5

TT1 represents the time (in seconds) to first integer solution
TTB represents the time (in seconds) to best integer solution
IPBEST is the best integer solution value found
IPOPT is the optimal integer solution value

Table 2: Results of the GPSIMAN and MIPSA codes on PZIP models

Comparing the results presented in Table 2 shows that MIPSA outperformed the original GPSIMAN code. Whilst MIPSA could find at least one feasible solution in all the PZIP models, the original GPSIMAN code fails to obtain a feasible solution in some of the models, namely: *cap6000, mitre, nw04, p0282* and *p2756*. In one further model, *mod010*, the best solution found by MIPSA had a value of 6663 and was found after 4872.3 seconds whereas the best solution found by GPSIMAN had a value of 6665, found after 954.34 seconds. However, since both approaches ran for the full time limit of 6000 seconds, MIPSA may be considered to have outperformed GPSIMAN on this model by virtue of this better solution. Finally, it can be seen that the 'time to first integer feasible solution' for MIPSA is, in general, better than in the original GPSIMAN code, justifying the use of the B&B depth first approach for finding the intitial feasible solution.

Another set of experiments was set up with a view to comparing the performance of PACO with the standard implementation of B&B in the FortMP MIP [8] solver and the results are presented in Table 3.

		B&B		**PACO**	
Name	**IPOPT**	**IPBEST**	**TTB**	**IPBEST**	**TTB**
air03	340160	340160	1000.2	340160	527.5
air04	56137	56137	1453.4	56137	798.8
air05	26374	26374	360.74	26374	103.2
cap6000	-2451377	-2423920	745.34	-2451377	203.1
l152lav	4722	4742	360.74	4722	230.2
lseu	1120	1120	71.2	1120	70.9
mitre	115155	125115	3843.3	124065	366.2
mod008	307	307	38.05	307	35.05
mod010	6548	6653	360.06	3600.65	1200.3
nw04	16862	16862	254.84	16862	156.8
p0033	3089	3089	3.08	3089	1.35
p0201	7615	7615	34.98	7615	34.95
p0282	258411	258411	1016.4	258411	818.6
p0548	8691	9682	1984.5	8691	897.6
p2756	3124	3870	4759.1	3721	3098.3
stein27	28	28	14.2	28	10.0
stein45	30	30	343.5	30	134.2

TTB represents the time (in seconds) to Best integer solution
IPBEST is the best integer solution value found
IPOPT is the optimal integer solution value

Table 3: Comparison between B&B and PACO results on PZIP models

We consider PACO to be a parallel heuristic, akin to stage 1 of the PB&B algorithm described in [19] in which alternative B&B trees are run concurrently to process MIP models. One measure of the effectiveness of this parallel approach is *relative speed-up* with respect to the individual approaches (B&B and SA).
We define

$$\mu_{B\&B} = \frac{TTB(B\&B)}{TTB(PACO)},$$

and

$$\mu_{MIPSA} = \frac{TTB(MIPSA)}{TTB(PACO)},$$

to be the two speed-up measures, where $TTB(MIPSA)$ and $TTB(B\&B)$ represent the time taken for the independent MIPSA and B&B procedures to find the best integer feasible solutions. $TTB(PACO)$ is the corresponding time for the parallel co-operative algorithm. In computing the speed-up measures, we assume that the best objective function value of the co-operative algorithm is *at least as good* as the best objective function value obtained using MIPSA and/or B&B independently. In Table 4 we present the values of μ_{MIPSA} and $\mu_{B\&B}$ for the models considered.

Since $TTB(PACO)$ is common, it can be seen that $\mu_{MIPSA} < \mu_{B\&B}$ represents cases in which the serial MIPSA procedure performs better than the serial B&B algorithm. Similarly, $\mu_{B\&B} < \mu_{MIPSA}$ represents the opposite scenario. Since two processors are involved in the co-operative parallel heuristic, a value of $\mu_{B\&B} > 2$ or $\mu_{MIPSA} > 2$ indicates a super-linear speed-up with respect to the B&B or MIPSA execution.

In these experiments, the co-operative approach did not produce a worse solution than produced by either of the two independent approaches. On four models (*cap6000, mitre, mod010* and *p2756*), the co-operative approach produced an improved solution compared to that produced by both the independent approaches. The values of $\mu_{B\&B}$ and μ_{MIPSA} presented in Table 4 show considerable variation. With the exception of *air04* when processed by MIPSA, all speed-up measures have a value of at least 1, with most occurences having a value greater than 2. This empirical evidence on both solution quality and execution time indicates a generally superior performance for the co-operative heuristic compared to the two independent approaches.

MODEL NAME	μ_{MIPSA}	$\mu_{B\&B}$
air03	1.31	1.90
air04	0.94	3.43
air05	3.76	3.50
*cap6000	7.66	3.67
l152lav	3.33	1.57
lseu	12.44	1.00
*mitre	2.39	10.50
mod008	1.15	1.00
*mod010	4.06	3.00
nw04	6.30	1.57
p0033	4.39	2.28
p0201	1.57	1.00
p0282	2.37	1.24
p0548	4.83	2.21
*p2756	1.76	1.54
stein27	1.67	1.42
stein45	1.12	2.56

* indicates models where the co-operative approach found better solutions than either of the independent approaches

Table 4: Relative Speed-up values of the parallel co-operative heuristic (PACO) on PZIP models

A conclusion drawn from these investigations is that there is a symbiotic relationship between MIPSA and B&B which is exploited through exchanging bounds via PVM. B&B provides good starting points for local searches to be carried out by MIPSA while MIPSA in turn can provide better bounds to guide the B&B search.

4.4.2 Summary of Results for MZIP Models

A similar set of experiments to those carried out for PZIP models was performed on a set of MZIP models described in Table 5, from the library of MIP benchmarks [1]. These results in Tables 6–8, are shown in the same order and format as Tables 2–4 for the PZIP models.

Model Name	No. of constraints	No. of binary variables	No. of continuous variables	Total no. of variables	No. of non zeros
10Teams	230	1800	225	2025	14175
Bell3A	123	39	94	133	441
Blend2	274	231	122	353	1497
DanO3MIP	3202	552	13321	13873	79656
DCMulti	290	75	473	548	1833
EGOUT	98	55	86	141	392
FIBER	363	1254	44	1298	4298
GESA2	1392	240	984	1224	6000
Markshare1	6	50	12	62	324
MISC07	212	259	1	260	8620
QNET1	503	1288	253	1541	4746
Rentacar	6803	55	9502	9557	42019
Swath	884	6724	81	6805	34966

Table 5: Model statistics of MIPLIB MZIP models

Name	**IPOPT**	**IPBEST**	**TT1**	**TTB**
10Teams	924	924	342.78	761.35
Bell3A	878430.32	N/A	N/A	N/A
Blend2	7.598985	7.598985	42.43	49.31
Dan03MIP	728.1[†]	N/A	N/A	
DCMulti	188182	188122	50.92	106.23
EGOUT	568.101	568.101	10.34	47.98
FIBRE	405935.18	414548.63	2838.40	6540.11
GESA2	25821263	25785278.0	100.23	5698.34
Markshare1	1	8	202.39	6923.40
MISC07	2810	2810	56.34	80.76
QNET1	16029.6927	17576.181	39.87	648.49
Rentacar	30356761	30356761	12.40	17.28
Swath	497.603	588.77401	1501.67	3289.51

TT1 represents the time (in seconds) to first integer solution
TTB represents the time (in seconds) to best integer solution
[†] means "Not proven as optimal"

Table 6: Results of MIPSA code on MZIP models

		B&B		PACO	
Name	**IPOPT**	**IPBEST**	**TTB**	**IPBEST**	**TTB**
10Teams	924	924	335.52	924	245.64
Bell3A	878430.32	1481630.5	64.23	878430.32	64.33
Blend2	7.598985	7.598985	165.43	7.598985	53.41
Dan03MIP	728.1†	755.701	5241.74	755.701	3232.11
DCMulti	188182	188122	40.24	188122	32.28
EGOUT	568.101	568.101	5.11	568.101	5.10
FIBRE	405935.18	414548.63	1337.91	405935.18	897.94
GESA2	25779856.4	25785278.0	698.34	25779856.4	126.66
Markshare1	8	1	203.85	1	203.83
MISC07	2810	2810	24.71	2810	24.73
QNET1	16029.6927	17576.181	71.69	17576.181	53.74
Rentacar	30356761	30356761	11.47	30356761	10.44
Swath	497.603	560.94781	1421.79	560.94781	

TTB represents the time (in seconds) to best integer solution
IPBEST is the best integer solution value found
IPOPT is the optimal (or best known) integer solution value
† means " Not proven as optimal"

Table 7: Comparison between B&B and PACO on MZIP models

The extensions we have made to GPSIMAN have enabled MIPSA to successfully process MZIP models. In general, the MIPSA code performed better on models in which the ratio of binary variables to continuous variables is high. Moreover, the effectiveness of the algorithm depends to a large extent on how quickly the depth-first B&B provides integer solutions about which a local search is performed. This feature becomes even more evident in cases where the SA code frequently encounters infeasibilities late-on in the run. In such cases, the time taken by successive B&B invocations dominates the time of the other operations within the SA framework.

MODEL NAME	μ_{MIPSA}	$\mu_{B\&B}$
10Teams	3.10	1.37
Bell3A	N/A	1.00
Blend2	0.92	3.10
Dan03MIP	N/A	1.62
DCMulti	3.29	1.25
EGOUT	9.41	1.00
*FIBRE	7.28	1.49
*GESA2	44.99	5.51
Markshare1	1.00	1.00
MISC07	2.28	1.00
QNET	0.74	1.33
Rentacar	1.66	1.10
Swath	2.31	1.10

* indicates models where the co-operative approach found better solutions than either of the independent approaches

Table 8: Relative Speed-up values of the parallel co-operative heuristic (PACO) on MZIP models

As in the case with PZIP models, the values of μ_{MIPSA} and $\mu_{B\&B}$ are generally greater than one thereby emphasizing the usefulness of the co-operative approach. The results of the parallel heuristic on MZIP models are less impressive than those on PZIP models. In two of the models (*FIBRE* and *GESA2*), the parallel heuristic found better solutions than either of the two independent approaches. In both cases, the solutions found were proven optimal solutions.

5 Summary and conclusions

In this paper, we have investigated the use of the simulated annealing meta-heuristic for solving integer programming models. Our implementation is based on the GPSIMAN [4] heuristic of Connolly. However, the enhancements that we have incorporated within GPSIMAN makes our implementation more versatile as it can process a wider class of problems, namely MZIP. In addition, our implementation, based on a feasibility restoration scheme, outperforms GPSIMAN on the test problems considered. We also describe a co-operative algorithm which provides a novel way of combining increasingly popular heuristics and metaheuristics with the exact B&B algorithm. The co-operative algorithm is implemented on a parallel platform and our experimental results indicate the superior performance of this scheme.

References

[1] R.E. Bixby, S. Ceria, C.M. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[2] R.E. Bixby, W. Cook, A. Cox, and E.K. Lee. Parallel mixed integer programming. *Research Monograph CRPC-TR95554, Center for Research on Parallel Computation, Rice University*, 1995.

[3] N.E. Collins, R.W. Eglese, and B.L. Golden. Simulated annealing – an annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8:205–307, 1988.

[4] D. Connolly. General purpose simulated annealing. *Journal of the Operational Research Society*, 43(5):495–505, 1992.

[5] K. Darby-Dowman and J. Little. Properties of some combinatorial optimization problems and their effect on the performance of integer programming and constraint logic programming. *INFORMS Journal on Computing*, 10(3):276–286, 1998.

[6] K.A. Dowsland. Simulated annealing. In C.R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 20–69. McGraw-Hill, 1995.

[7] R.W. Eglese. Simulated annealing: a tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[8] E.F.D. Ellison, M. Hajian, R. Levkovitz, I. Maros, and G. Mitra. A Fortran based mathematical programming system: FORTMP. *Brunel University, London and NAG Ltd, Oxford*, 1995.

[9] A.P. French, A.C. Robinson, and J.M Wilson. A hybrid genetic/branch and bound algorithm for integer programming. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 238–240. Springer-Verlag, Vienna, 1998.

[10] E.L. Johnson, G.L. Nemhauser, and M.W.P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[11] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598).

[12] R.S. Laundy. Implementation of branch and bound algorithms in XPRESS-MP. In *Operational Research in Industry*, pages 25–41. Macmillan Press, 1999.

[13] F.H. Lee and G.S. Stiles. Parallel simulated annealing: several approaches. In J.A. Board, editor, *Transputer Research and Applications 2, IOS Press, Amsterdam*, Transputer and OCCAM Engineering Series, pages 201–214. 1990.

[14] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[15] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34:111–124, 1986.

[16] G. Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155–170, 1973.

[17] G. Mitra, I. Hai, and M.T. Haijan. A distributed processing algorithm for solving integer programs using a cluster of workstations. *Parallel Computing*, 23(6):733–753, 1997.

[18] P Moscato. An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Annals of Operations Research*, 41:85–121, 1993.

[19] V.L. Nwana. *Parallel Algorthms for Solving Mixed Integer Linear Programs.* PhD thesis, Brunel University, Department of Mathematical Sciences, Uxbridge, Middlesex, UB8 3PH, UK, November 2001.

[20] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill, 1995.

[21] C.R. Reeves and C. Hohn. Integrating local search into genetic algorithms. In *Modern Heuristic Search Methods.* John Wiley and Sons Ltd., 1996.

[22] J. Teghem, M Pirlot, and C Antoniadis. Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem. *Journal of Computational and Applied Mathematics*, 64:91–102, 1995.