Control and Cybernetics

vol. **29** (2000) No. 3

Integrating pivot based search with branch and bound for binary MIPs

by

Arne Løkketangen* and David L. Woodruff**

*Molde College
Britveien 2, 6411 Molde, Norway
E-mail: Arne.Lokketangen@hiMolde.no

**Graduate School of Management, UC Davis
Davis, CA 95616, U.S.A.
E-mail: dlwoodruff@ucdavis.edu

Abstract: This paper examines integration of a sophisticated, pivot-based tabu search into branch and bound for 0-1 MIPS and global diversification tests using chunking. Issues related to behavior of a tabu search within a branch and bound algorithm are analyzed using computational experiments. Results are presented showing that the inclusion of the local search sometimes results in fewer nodes and lower CPU times even when used in a callback mode. The main benefit in incorporating a pivot based heuristic is that an integer feasible solution can be found earlier in the branching process. Computational experiments are presented showing that for some instances the overall search time is improved, while for some others the tabu search can find good solutions quickly.

Keywords: tabu search, branch and bound, chunking, heuristics.

1. Introduction

Linear programming models with a mixture of real-valued and binary variables are often appropriate in strategic planning, production planning with significant setup times, personnel scheduling, and a host of other applications. The abstract formulation for linear problems with binary integers takes as data a row vector c of length n, an $m \times n$ matrix A and a column vector b of length m. Let D be the index set $1, \ldots, n$. The problem is to select a column vector, x, of length n so as to

subject to

$$\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{1}$$

$$x_i \in \{0,1\},\ i \in I \tag{2}$$

$$x_i \geq 0,\ i \in D \setminus I, \tag{3}$$

where the index set I gives the variables that must take on zero-one values. We use ℓ to indicate the number of integer variables.

Meta-heuristics, or heuristic search (HS) methods, such as tabu search (Glover and Laguna, 1997) have also been proposed for such problems and have been very successful in some settings. HS methods are often disadvantaged because they typically cannot, by themselves, prove optimality of their solutions (exceptions include Glover, 1996). A related problem is that it is difficult to know when the search should be terminated.

Exact methods of solving these problems such as branch and bound or branch and cut operate by fixing some of the integer variables at an integer value; each set of fixed integer values defines a node in the search tree. The remaining integer values are left free to take on real or integer values; the resulting problem is referred to as the *relaxation* for the node. Exact methods progress from node to node to implicitly exhaust all possible combinations of integer values. The best integer feasible solution currently available can be used to bound out some combinations.

In this paper, issues related to behavior of a tabu search within a branch and bound algorithm are analyzed using computational experiments. We integrate a pivot based search (Aboudi and Jörnsten, 1994; Glover and Løkketangen, 1997; Løkketangen and Glover, 1996; Løkketangen, Jörnsten and Storøy, 1994) with a branch and bound algorithm. The integration takes place primarily in the form of searches launched from the nodes of the branch and bound tree. These searches are terminated when an integer feasible solution is found or after some number of pivots, NI. Any time a new best is found, the search is continued for an additional NI of pivots. Chunking (Woodruff, 1998) is used to detect solutions that should be used for special searches that begin at the LP relaxation and to determine when the use of pivot searches should be discontinued.

Exact methods sometimes include 'quick' heuristics to help solve sub-problems and/or improve bounds. For example, MINTO (Savelsbergh and Nemhauser, 1996) uses what they call a 'diving' heuristic every 25 nodes to see if an integer feasible solution can be obtained using the solution for the node's relaxation as a starting point. An algorithm that combines genetic algorithms with branch and bound for satisfiability problems has been proposed by researchers at Loughborough University (French, Robinson and Wilson, 1997). They are also pursuing other forms of integration between local search and branch and bound as part of an on-going research project.

The next section describes the pivot based search and its behavior when

establish local and global search parameter values are described. In Section 3 global issues are discussed such as termination of the use of pivot searches. The full algorithm is summarized and computational experiments are described in Section 4. The paper closes with a section devoted to conclusions and directions for further research.

2. Searching within branch and bound

2.1. Pivot based tabu search

The branch and bound B&B algorithm (BBA) calls back to the controlling metaheuristic level at each new node in the B&B tree, and a local search from the current node might be launched (see Section 3). This local search is based on extreme point pivoting in a bounded variable simplex algorithm. A short description of the search is outlined in the Appendix, while the reader is referred to Glover and Løkketangen (1997), Løkketangen and Glover (1996), for extensions and variants.

As the search is launched from nodes in a B&B tree, there are some special considerations that come into play, setting this use of the pivot based search somewhat apart from other implementations. First, there is no need to incorporate any diversification or intensification schemes into the local search, as these matters are addressed by the chunking mechanism (see Section 3.1) and the path-relinking based target searches (see Section 3.2), respectively. Second, the purpose, or focus, of the search is somewhat different from the stand-alone search, in that for some of the searches, the emphasis is shifted more towards obtaining integer feasibility quickly. This focus is controlled by the parameter SKEW.

Another issue is that all the searches that are launched are similar, in that they are all variants of the same original problem, but with a varying number of integer variables fixed at the different nodes of the B&B tree. No variables are fixed at the root node, while progressively more integer variables are fixed further down the tree. This implies that searches should be launched only from nodes "sufficiently" apart (see Section 3).

One problem with the pivot based search is that the neighborhood is very costly to evaluate. Large speedups can be obtained, however, by proper candidate list strategies. These strategies are linked to the notions of *Neighborhood Exploration* and *Neighborhood Exploitation*. A brief description of these measures are given below, while a fuller account can be found in Løkketangen and Glover (1998).

The full neighborhood in a pivot based search consists of the set of nonbasic variables. We choose to evaluate only part of the full neighborhood at each exploration phase. Typically, 10% to 30% of the possible neighbors are explored, controlled by the search parameter NFRAC. Random variable selection is used, as a round robin exploration scheme runs into trouble due to the fixed layout

The exploration phase gives a collection of candidate moves sorted in decreasing order of heuristic quality. After picking the best (non-tabu) move off the list, we might continue to exploit the remaining moves on the candidate list, thereby postponing the next (expensive) exploration phase. We thus proceed down the candidate list, stopping when encountering a bad move or when RTN moves have been explored. A bad move is defined as one that both reduces the degree of integer feasibility and results in a degradation in the objective function (see the Appendix). It should be noted that it is necessary to reevaluate the move evaluation and re-identify the leaving pivot variables for all moves picked off the candidate list after the first (non-tabu) move. Typical values for RTN are between 3 and 5.

2.2. Parameter settings for the searches

In order to use a pivot based search from the nodes of a BBA, we need to find parameter settings that are appropriate. This need, in turn, gives rise to a need to study the behavior of the pivot search when launched from nodes. Of course, the primary feature of searching in this environment is that some of the integer variables are fixed. Furthermore, they are not fixed in random patterns, but by using the rules and tricks built into a commercial MIP solver. The only way to study this behavior is by example. We select three instances for our study that have a varying fraction of their variables constrained to be integer and varying sizes. The instances for the behavior study are summarized in Table 1.

Name	Rows	Columns	Binary
misc05	300	136	74
misc06	820	1808	112
exdash	10	575	575

Table 1. Instances for node search behavior study

The first value to set is the skew factor, SKEW, for search for feasibility from a node. This task is rendered both difficult and easy by the fact that the search performance is extremely robust with respect to this parameter. It is difficult in the sense that it is hard to choose and easy in the sense that any choice is a good one.

To study the dependence of the search on this parameter we conducted a series of experiments using five replicates for each instance and parameter value, where each replicate is characterized by a different random number stream. For each replicate, we launched a search from every node with the experiment ending at branch and bound termination or after 100 searches. Each search was

primary goal in such searches is the discovery of an integer feasible solution. Table 2 summarizes the results. The results are pro-intuitive in that lower skew factors generally result in a few more feasible solutions, but not significantly. The objective function values obtained were also of no help in picking a parameter value. There was not a statistically significant performance difference. Due to the condition of the search finding a feasible solution, the number of iterations required did not vary significantly with the parameter for misc05 or misc06, but for exdash, fewer iterations were required with SKEW = 0.01 (an average of 189 versus averages of very nearly 220 for the other settings). This difference is not big, but lacking any other clear criteria, we use it to set the value of SKEW at 0.01 for searches from a node.

	SKEW			
Instance	1	0.1	0.01	0.001
misc05	0.56	0.53	0.56	0.60
misc06	1.00	1.00	1.00	1.00
exdash	0.43	0.53	0.55	0.56

Table 2. Fraction of searches from nodes resulting in integer feasibility

The next two parameters are somewhat more complicated. They are referred to collectively as the *neighborhood parameters*. In order to set the values of NFRAC and RTN, we conduct a study with samples of searches from the nodes. For each of the three instances, we again conduct an experiment with five replicates for each parameter setting. We search from a sample of the nodes with roughly seven searches per replicate. The searches were terminated at integer feasibility or after n pivots. Table 3 summarizes the results. The columns

		misc05			misc06			exdash		
		Ave.	Frac.	Cond.	Ave.	Frac.	Cond.	Ave.	Frac.	Cond.
NFRAC	RTN	Time	Feas.	Obj.	Time	Feas.	Obj.	Time	Feas.	Obj.
0.1	1	1.3	0.48	2997	15.4	1.00	12930	23.4	0.63	−708528
	3	1.3	0.48	2997	10.0	1.00	12977	16.1	0.51	−698337
	5	1.3	0.48	2997	8.4	1.00	12974	14.1	0.51	−713648
0.2	1	1.8	0.40	2988	22.6	1.00	12939	32.2	0.80	−717328
	3	1.5	0.52	2997	12.3	1.00	12917	29.3	0.63	−714330
	5	1.5	0.52	2997	9.3	1.00	12918	20.0	0.80	−698043
0.3	1	2.0	0.60	3010	33.8	1.00	12906	42.5	0.77	−722025
	3	1.7	0.56	2996	13.8	1.00	12968	35.1	0.86	−702118
	5	1.7	0.56	2996	14.0	1.00	12915	31.2	0.66	−718580

labeled "Ave. Time" give the average number of seconds for each search from the node (on a computer with a 120 MHz Pentium Processor), "Frac. Feas." is the fraction of searches that resulted in a feasible solution, and "Cond. Obj." gives the average objective function value for those searches that resulted in integer feasibility.

There are no parameter values that are clearly superior, so we resort to heuristic reasoning. The product of the Ave Time and Frac Feas columns is the expected amount of time per feasible solution. This would be a good thing to minimize, all other things being equal. This criterion suggests that the NFRAC parameter be set at 0.1, with 3 being a reasonable value for RTN; however, there are objective function considerations. An NFRAC value of 0.2 would seem better in terms of getting a lower objective function value (however, the pairwise differences are typically significant only at levels below 0.9). We select two sets of neighborhood parameter pairs (NFRAC, RTN): the "slower" pair $(0.2, 3)$ and the "faster" pair $(0.1, 3)$. For the search that is launched from the root node of the tree, we use the slower pair based on the heuristic reasoning that a quality solution is preferred and worth the wait. For subsequent nodes, we use the faster pair when a larger number of iterations are allocated and the slower pair with fewer iterations based on the typical heuristic reasoning that we can "afford more time" for each move when there will be fewer moves and furthermore that we need to "make the most" of each move. When there are possibly more moves, we cannot "afford to spend a lot of time on each move."

To set the number of iterations, NI, we need to pause to consider the nature of integration of a pivot search into a branch and bound algorithm. A BBA does not select variables to fix at random. The B&B tree is searched in a manner that incorporates heuristic variable selection and node selection based on the ability to solve the resulting subproblems efficiently and effectively. It would not be prudent to launch a search from every node because it is very likely that there is not much difference from one node to the next. To provide a high likelihood that we are launching searches from nodes that are significantly different, we search every ℓ nodes.

In order to determine an appropriate number of pivots (or moves) for each search, NI, we again turn to experimental analysis of search behavior. Figures 1 through 3 display the results of an experiment with five replicates of samples from the nodes of the exdash instance. The searches were terminated at integer feasibility or when n pivots had been executed without finding an integer feasible solution (there is no guarantee that an integer feasible solution exists for the values fixed at each node). The graphs show one point for each search (with many points overlapping) giving the number of pivots versus the fraction of the integers that have been fixed by the branch and bound algorithm. The main thing to notice is that the number of pivots goes down with the fraction fixed as would be expected, but only sightly.

Since our initial goal in searching from a node is to see if there is an integer

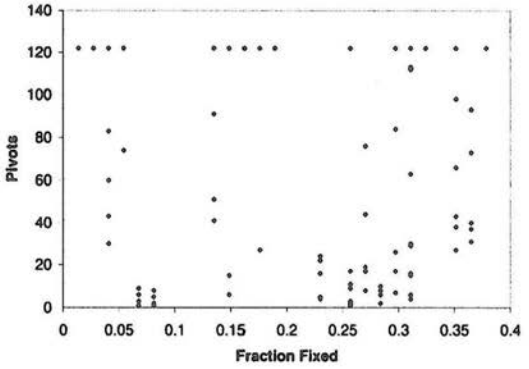


Figure 1. Pivots to integer feasibility for misc05 (stopped at n)

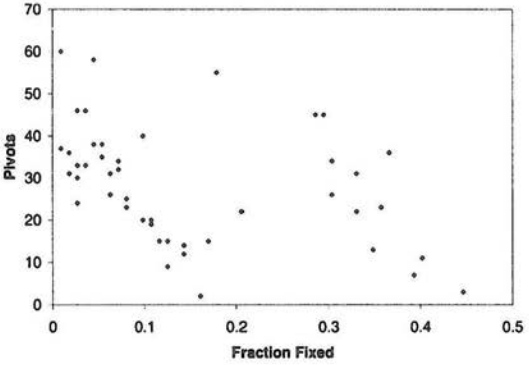
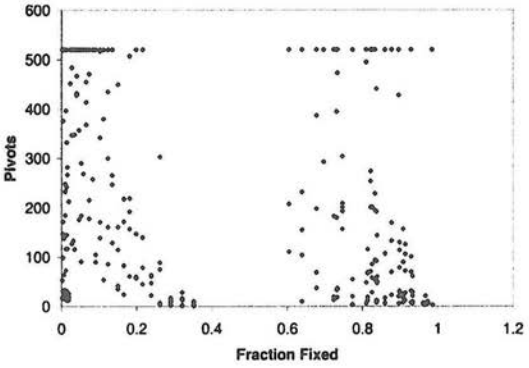


Figure 2. Pivots to integer feasibility for misc06 (stopped at n)



goal is to minimize the number of pivots per integer feasible solution found. We should set the number of iterations, NI, accordingly. For any given instance, we cannot know *a priori* what that value would be, so we set $\text{NI} = \ell$ for the first γ searches and then determine the value of NI that would have resulted in the minimum number of pivots per feasible solution. For the remaining searches we set NI to be the minimum of this value and $n/10$. Assuming that this value is lower than ℓ, it might seem that we will have "wasted" pivots during the first γ searches, but this probably is not really the case. We are not only sampling to determine a good value for NI, because these searches are being done early in the BBA process, where a good bound will have the greatest affect on pruning the tree. Hence, longer searches are in order for the first few nodes sampled.

To clarify the search for NI, we formalize it. Let λ_i' be the sup of the number of iterations needed to find an integer feasible solution during search i. Let $\lambda_i(\text{NI})$ be the number of iterations expended in search number i. So, $\lambda_i(\text{NI}) = \min(\lambda_i', \text{NI})$ for finite NI. If we let δ be the usual indicator function, we can write down our objective. After the first γ searches, we want the value NI that is

$$\underset{\text{NI}}{\operatorname{argmin}} \sum_{i=1}^{\gamma} \frac{\lambda_i(\text{NI})}{\delta(\text{NI} \leq \lambda_i')} \tag{4}$$

based on the heuristic justification that this will approximately result in the lowest possible number of pivots per feasible solution found during subsequent searches. We state again a lemma that has been given and proven in a wide variety of settings.

Lemma 1 *If a search is conducted with* $\text{NI} = \ell$ *for the first* γ *nodes, then the minimum for Equation 4 subject to* $\text{NI} \leq \ell$ *is achieved at one of the observed values,* $\lambda_i(n)$, $i = 1, \ldots, \gamma$.

This lemma leads immediately to a proof of optimality for an algorithm that checks at most γ candidate values for NI. We use the notation NI* to refer to the value found by such an algorithm with a constraint that the value be at least $n/10$. In our implementations we always use $\gamma = 5$ based on the argument that this is the minimum reasonable size for any statistical sample.

3. Global issues

In the previous sections we have described the pivot based search and experiments done to establish parameter values for searches launched from nodes. This leaves us with a few global issues unresolved. Two related questions must be answered. When should the pivot based searches be terminated? When should special searches be launched? Both of these questions can be addressed via the use of chunking.

A major advantage of BBA is that it will eventually result in proof of opti-

the optimal solution has been found. The relaxations must be solved so that branches of the tree can be fathomed, but continued use of a pivot search is clearly pointless. We cannot know for sure when the pivot searches should be terminated, but we can construct a mechanism that terminates them when they seem to be no longer operating from a diverse set of starting points.

Conversely, if we find a "unique" or "outlying" solution we might want to make use of it in a special search. The details of special searches are given in Section 3.2. The meaning of "unique" and "outlying" can be supplied by chunking.

3.1. Chunking

The basic idea behind chunking is to dramatically reduce the dimension of the solution vectors while retaining (or obtaining) a reasonable measure of distance between them. For pure binary problems, Hamming distances might suffice. However, for MIPs Hamming distances can be misleading because they do not take into account the real valued variables. One source of useful metrics is chunking. For a more general discussion of chunking, see Woodruff (1998, 1996).

A terse summary of chunking will be given here. We will refer to a non-empty subset of solution vector indexes as a *chunk*. An instance of a partial vector corresponding to a chunk is referred to as a *chunk instantiating value*. That is, a chunk instantiating value for a given chunk $C \subseteq D$ depends on the specific vector $\boldsymbol{x}$ whose values are under consideration. Let $\mathcal{P}$ be a partition of D with p cells. The mappings from chunk instantiating values to reals are referred to as *chunk valuation* functions, v. We will deal with a fairly specific valuation function. Let $v_{\mathcal{P}}(\boldsymbol{x})$ be the (column) vector of chunk valuations for a given vector $\boldsymbol{x}$ under partition $\mathcal{P}$. Each element of the valuation vector is defined as

$$v_C(\boldsymbol{x})) = \frac{1}{|C|} \sum_{i \in C} x_i,$$

where $|C|$ is the number of indexes in the chunk for each chunk $C \in \mathcal{P}$. If we index the chunks $C \in \mathcal{P}$, we can write v_i in the same way for $i = 1, \ldots, p$.

We can use chunking to quantify the location and shape of a group of solution vectors. This characterization then allows us to quantify the distance between solution vectors using the shape to define a metric. In particular, we measure the distance to the location of the group used to define the shape.

To get the appropriate metric, we make use of a location vector (mean) μ of length p and a $p \times p$ shape matrix (covariance matrix) S. A p-vector v can be said to have a *Mahalanobis distance* from μ using S that is given by

$$d^2(\mu, S; v) = (v - \mu)^T S^{-1} (v - \mu). \tag{5}$$

Short distances are consistent with μ and S and longer distances are not as

The motivation for this metric comes from multivariate probability theory. If S and μ define a multivariate normal distribution, then for points governed by the distribution, the Mahalanobis distances are distributed χ^2_p (see e.g., Anderson, 1984, pages 72–75). We make use of a matrix S that is the covariance estimate for the valuation vectors of the most recent N feasible integer solutions sampled. In our work described here, p is left as a free parameter and N is fixed at $5p$. We will make use of the χ^2_p distribution only as a method of characterizing distances in a way that depends on the data and p in a rational way that can be characterized by the single parameter α; hence we really are not relying at all heavily on normality.

If for the most recent N integer feasible solutions that have been obtained S lies entirely on a hyperplane of dimension p, the pivot searches are terminated. This corresponds to a lack of diversity in the results of the pivot searches and/or BBA thereby indicating that further pivot searches will be a waste.

If solutions, x, are encountered whose valuation vectors $v_C(x)$ are far from the mean of the sample of N solutions, then a special search is indicated. The notion of "far" is captured by a parameter α that gives the threshold for the inverse of the chi^2_p distribution for $d^2(\mu, S; v_C(x))$. This parameter typically has a low value; e.g., of the order of one half. The special searches that are launched are now described.

3.2. Special searches

After a best-so-far solution x^* has been found, the chunking mechanisms try to identify distant (w.r.t. the current sample of solutions) solutions, x'. When such a distant solution has been identified, a *2-target* search is launched. This is a variant of path relinking, with the purpose of launching a new search into unknown territory, while at the same time keeping good parts of the solutions. To put it another way: this will intensify the search, while at the same time diversifying the search.

The starting point of this search is the relaxed root node solution $LP*$ (being an upper bound on the objective function value), and the target for the path relinking is the hyperplane defined by the common integer solution values of x^* and x'. All integer variables are freed.

To allow the search to focus on this hyperplane, the integer infeasibility part of the move evaluation function is temporarily modified. (The objective function value component is unaltered, as is the aspiration criterion — see the Appendix). Instead of using the normal integer infeasibility measure of summing up over all the integer variables the distance to the nearest integer, we use the following scheme:

- Sum up over all the integer variables.
- If the two targets have the same integer solution value for the variable,

- If the two targets differ, use the normal integer infeasibility measure (i.e. the closest integer value).

When the search reaches the hyperplane connecting x^* and x', the normal move evaluation function is reinstated, and the search continues in normal fashion for NI iterations.

4. Computational results

4.1. Summary of the algorithm and implementation

The searches from nodes are implemented as callbacks from the branch and bound code XPRESS-MP version 10.1 (Dash Assoc., 1998) with default parameter settings. A special pivoting interface is implemented to facilitate the pivot search. Termination of further pivot searches is implemented simply by canceling the callbacks. The code for the pivot based search is implemented in C++ and experiments were conducted on a 200 Mhz Pentium PC.

Name	SKEW	NFRAC	RTN	NI
ROOT	0.10	0.2	3	n
SMALL	0.01	0.1	3	ℓ
BIG	0.01	0.2	5	NI*

Table 4. Parameter sets

To describe the algorithm it is useful to refer to the parameter sets given in Table 4. The algorithm proceeds in four phases, but when a new best solution is found during any phase a search from that solution is launched for NI iterations with the variables that were fixed at that point in time and using the same parameter set as the search from the node except with a SKEW value one order of magnitude greater and termination on the iteration limit or a new best found. For every search from a node, the search is terminated at the iteration limit or when an integer feasible solution is found. Special searches have other termination criteria. Here are the phases:

1. Search from the root node (the LP relaxation) using the parameter set named ROOT.
2. For the next γn nodes, search from every n^{th} node using the parameter set named SMALL. Calculate the value of NI that would have minimized Equation 4 for the γ searches done so far. Use this value for NI in subsequent node searches.
3. Continue node searches from every n^{th} node using the parameter set named BIG until $N = 5p$ integer feasible solutions have been found. Construct valuation vectors for each of the N solutions and the resulting

4. Continue launching searches from every n^{th} node using parameter set BIG. Launch a special search as described in Section 3.2 if a solution, $\boldsymbol{x}$, is encountered such that its valuation vector $\boldsymbol{v}_C(\boldsymbol{x})$ has the property that $d^2(\boldsymbol{\mu}, \boldsymbol{S}; \boldsymbol{v}_C(\boldsymbol{x}))$ is greater than the inverse of the χ_p^2 distribution at 0.5. Terminate callbacks if the determinant of $\boldsymbol{S}$ becomes zero or if the sample is refreshed without a special search launched.

4.2. Results

The computational tests were done on a set of test problems obtained from miplib (ftp://softlib.cs.rice.edu/softlib/miplib/) and Dash Associates (exdash and hpw15) (see address at Dash Assoc., 1998). These are listed in Table 5, together with the problem dimensions. Some of these are not too difficult, while others require substantial amounts of cpu time to prove optimality. The selection contains both pure IP's and MIP's.

Name	Rows	Columns	Binary
dcmulti	290	548	75
egout	98	141	55
exdash	10	575	575
hpw15	55	90	30
misc05	300	136	74
misc06	820	1808	112
misc07	212	260	259
pk1	45	86	55
pp08a	136	240	64
pp08aCUTS	246	240	64
vpm1	234	378	168
vpm2	234	378	168

Table 5. Test cases

It should be pointed out that our heuristics are "grafted" onto Xpress. This implies that the data structures that are internal to Xpress have to be copied over to our local data structures. This overhead, together with the overhead associated with the actual callbacks, is quite significant, and leads to less favorable comparisons.

The main benefit for the B&B of the local search is to hopefully obtain integer feasible solutions earlier, and thus better cut-off values, leading to fewer nodes expanded and shorter overall search time. It should be noted, however, that better cut-off values does not necessarily speed up the B&B, neither in

Due to the chunking criteria, local searches usually are not used after a few thousand nodes. This has the beneficial effect of not wasting time on the local searches for the test cases that need many nodes in the B&B.

A series of tests, with a maximum of one million nodes for the B&B, were run for each of the test cases. For each test case the B&B were run alone, as well as 5 runs (with different random seeds) with full search and 5 runs with root node search only.

For many test cases there was no appreciable effect, even though integer feasible solutions were found quite early. Table 6 shows the overall running time for the B&B on its own, as well as the average *extra* time spent when adding the local searches. The column **Root** signifies root-only search, and the **Full** column the searches as described in the previous chapter. A bold figure in the **B&B** column signifies that the optimum was found. For dcmulti and misc06 the extra search time is quite large, while for the rest it is quite negligible.

Name	B&B	Root Only	Full
dcmulti	**29**	37.6	107
exdash	**15**	3.5	29
misc05	**15**	3.5	29
misc06	**10**	99.1	173

Table 6. Times for test cases where local search has a bad effect

There are also many instances for which the search at the root node is able to find quickly a good, feasible solution, but this does not necessarily result in a speedup for the overall combined algorithm. These instances are summarized in Table 7. The column labeled "Root Only Gap" gives the percent deviation between the solution found by the root node tabu search and the solution found by the branch and bound algorithm. The quality of the solutions is reasonable given that the time required is very low. It is also important to note that the time

Name	B&B Time	Root Only Time	Root Only Gap	Full Time
misc07	**1,009**	15.7	4%	1,199
pk1	**19,705**	1.6	18%	20,225
pp08a	28,263	4.5	27%	28,208
pp08aCUTS	38,825	7.0	12%	39,565
vpm1	**34,401**	4.4	5%	34,414
vpm2	34,813	7.6	12%	35,273

Table 7. Test cases where a root node search was useful and the Full Time was

for the full search is almost the same as the time for B&B alone. In the interest of being conservative in drawing conclusions, we cannot claim that this "proves" that our methods would speed up a branch and bound algorithm for some instances if incorporated by the professionals who develop commercial software with full access to internal data structures. However, given the overhead of implementing as a callback, this conclusion is strongly supported. Certainly, these results suggest that the full search is worthy of further consideration.

Table 8 shows the search times for the test cases where the local searches have good effect. The columns **Root** and **Full** now give the total search time including the searches (as opposed to the extra search time as shown in Table 6). The reason for the same time for both **Root** and **Full** search for hpw15 is that the search is so short that no searches are launched after the first (root node) search. The instance slsp3 is the deterministic equivalent of a stochastic mixed integer programming instance that is available at www.gsm.ucdavis.edu/~dlw/testcase-ejor.zip on the web. These instances are quite "easy" in the sense that they can be solved quickly with, or without, the addition of local search. However, the speedup that we can offer can be significant when such problems are solved as sub-problems for a larger problem (for example, when using progressive hedging for stochastic programming, Rockafellar and Wets, 1991.) Note that the reduction in full search times versus B&B alone is a very large percentage, so if such problems must be solved repeatedly the overall savings can be quite meaningful.

Name	B&B	Root	Full
egout	**2.1**	**1.1**	**1.6**
hpw15	**2.4**	**1.0**	**1.0**
slsp3	**1.6**	**0.6**	**1.0**

Table 8. Times for test cases where local search has good effect

As can be seen, the best effect is for the root node search, less so for the full search. We believe that root node search can have a beneficial effect, and does not cost too much in terms of computational time. It might also be included for instances where the (proven) optimum might not be needed, but a good feasible solution suffices.

5. Conclusions and directions for further research

This paper has examined integration of a sophisticated, pivot-based local search into branch and bound for binary MIPS and global diversification tests using chunking. The search behavior is analyzed using computational experiments.

local search sometimes results in fewer nodes and lower CPU times even when used in a callback mode.

It is very difficult to beat a high quality commercial solver like Xpress from the "outside." Such code includes many "tricks of the trade" and has been refined over many years of programming. In addition, by using callbacks that must recreate data structures that are available internally we burden ourselves with additional overhead. However, in spite of this limitation the version that calls back to our code was fairly competitive with the commercial solver in terms of cpu use. Our experiments support the contention that tabu search embedded in branch and bound algorithm can be viable for some classes of instances.

The main benefit in incorporating a pivot based heuristic is the ability to find an integer feasible solution earlier in the branching process than one otherwise would. We were able to produce speedups on a few instances. These instances could be solved quickly anyway, but the fact that the speed improvement was significant could prove very valuable in a setting where many such instances must be solved as sub-problems or when a diverse set of solutions is needed (Glover, Løkketangen, and Woodruff, 2000). It remains for further research to characterize the classes of instances for which combinations of heuristic search with branch and bound are most useful.

Also left for future research are issues associated with incorporating population based tabu searches with branch and bound. The branch and bound process can generate a large number of candidate population members. In this paper we have demonstrated the usefulness of chunking for detection of diversity, so the ingredients are in place.

References

ABOUDI, R. and JÖRNSTEN, K. (1994) Tabu search for general zero-one integer programs using the pivot and complement heuristic. *ORSA J. Comput.*, **6**, 82–93.

Dash Assoc. (1998) *XPRESS-MP Reference Manual*, Version 10.1, Binswood Ave, Leamington Spa, Warwickshire, CV32 5TH, UK.

FRENCH, A.P., ROBINSON, A.C. and WILSON, J.M. (1997) Solving satisfiability problems using a hybrid genetic-algorithm/branch-and-bound approach. Working paper, Loughborough University, Loughborough, LE11 3TU, England.

GLOVER, F. (1996) Ghost Image Methods for Integer Programming. Working paper, University of Colorado, Boulder, CO, U.S.A.

GLOVER, F. and LAGUNA, M. (1997) *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

GLOVER, F. and LØKKETANGEN, A. (1997) Solving zero-one mixed integer

GLOVER, F., LØKKETANGEN, A. and WOODRUFF, D.L. (2000) Scatter search to generate diverse MIP solutions. With F. Glover and A. Løkketangen in *Computing Tools for Modeling Optimization and Simulation*, Laguna and Velarde, eds., 299–320, Kluwer.

LØKKETANGEN, A. and GLOVER, F. (1996) Probabilistic move selection in tabu search for zero-one mixed integer programming problems. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, eds., 555–570.

LØKKETANGEN, A. and GLOVER, F. (1998) Candidate list and exploration strategies for solving 0/1 MIP problems using a pivot neighborhood. In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voß, S. Martello, I.H. Osman and C. Roucairol, eds., Kluwer Academic Publishers, 141–155.

LØKKETANGEN, A., JÖRNSTEN, K. and STORØY, S. (1994) Tabu search within a pivot and complement framework. *Int. Trans. Oper. Res.*, **1**, 305–316.

Miplib. ftp://softlib.cs.rice.edu/softlib/miplib/

ROCKAFELLAR, R.T. and WETS, R.J.-B. (1991) Scenarios and policy aggregation in optimization under uncertainty. *Math. of OR*, **16**, 119–147.

SAVELSBERGH, M.W.P. and NEMHAUSER, G.L. (1996) *Functional Description of MINTO, a Mixed Integer Optimizer*, Version 2.3, ISYE, Georgia Institute of Technology, Atlanta, GA, 30332, U.S.A.

WOODRUFF, D.L. (1998) Proposals for chunking and tabu search. *EJOR*, **106**, 585–598.

WOODRUFF, D.L. (1996) Chunking applied to reactive tabu search. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, eds., 555–570.

Appendix — Pivot search details

The necessary background for the tabu search extreme point pivot search method is outlined here, using the notation and definitions of earlier studies. For more details, see Glover and Løkketangen (1997). The reader is assumed to be familiar with tabu search (see e.g. Glover and Laguna, 1997) and with the basic concepts of the bounded variable simplex method.

Given the 0-1 MIP problem formulation described in the introduction section, the first-level Tabu Search (TS) heuristic for this problem may then be outlined as follows:

Let $x*$ denote the best MIP feasible solution, and let $z*$ denote its objective function value.

1. Solve the LP relaxation (i.e. ignoring integer constraints) to obtain an optimal LP basic (extreme point) solution.
2. Consider the feasible pivot moves that lead to adjacent basic LP feasible solutions. If a candidate move would lead to an 0-1 MIP feasible solution x whose associated z value yields $z > z*$, record x as the new $x*$ and

3. Select the pivot move with the highest move evaluation, applying tabu restrictions and aspiration criteria.
4. Execute the selected pivot, updating the associated tabu search memory and guidance structures. Return to Step 2.

Note that the method may not necessarily visit the best MIP feasible neighbor of the current solution, since the move evaluation of Step 2 depends on other factors in addition to the objective function value (see below).

Neighborhood structure

Let $x(0)$ denote the current extreme point solution, let $\{x_j : j \in NB\}$ denote the current set of nonbasic variables, and let $\{x_j : j \in B\}$ denote the set of basic variables ($B = N - NB$). The extreme points adjacent to $x(0)$ have the form

$$x(h) = x(0) - D_h\theta_h \text{ for } h \in NB$$

where D_h is a vector associated with the nonbasic variable x_h, and θ_h is the change in the value of x_h that moves the current solution from $x(0)$ to $x(h)$ along their connecting edge.

We thus start the search from an integer infeasible point, and may also spend large parts of the search visiting integer infeasible solution states.

Move evaluation

The move evaluation function is composite, based on two independent measures. The first measure is the change in *objective function value* when going from $x(0)$ to $x(h)$, and the second measure is the change in *integer infeasibility*. Let $near(x_j(h))$ denote the integer nearest to the value $x_j(h)$. Then we define $u(h)$ to be the amount of integer infeasibility for the solution $x(h)$, where

$$u(h) = \sum_{j \in I} |x_j - near(x_j(h))|$$

Restricting consideration to $h \in NB$, we define

$$\Delta z(h) = z(h) - z(0)$$
$$\Delta u(h) = u(0) - u(h).$$

Note that it is not necessary to execute a pivot to identify $x(h)$ or the values of $u(h)$ and $z(h)$, since only the vector D_h, the scalar θ_h, and the current solution $x(0)$ are required to make this determination.

Move types and choice rules

The moves are grouped into four distinct move types, according to the values

Move Type	Defining Conditions
I	$\Delta z(h) < 0, \Delta u(h) > 0$
II	$\Delta z(h) > 0, \Delta u(h) < 0$
III	$\Delta z(h) \geq 0, \Delta u(h) \geq 0$
IV	all others

Table 9. Move types

The classification of move types is made according to the degree to which they may be considered favorable. Moves of type III are generally the most favorable, improving in both of the measures used in the move evaluation function, while moves of type IV are the least favorable, worsening in both measures. Moves of types I and II are improving in one of the measures, and worsening in the other.

The ranking of moves within each move group is as follows, with the best move in the group being indicated by Table 10.

Move Type	Move Ranking
I	$Max(\Delta z(h)/\Delta u(h))$
II	$Max(\Delta u(h)/\Delta z(h))$
III	$Max(\Delta u(h) * \Delta z(h))$
IV	$Min(\Delta u(h) * \Delta z(h))$

Table 10. Move classifications and ranking functions

When move evaluations tie within a group, we choose the move with the largest value of the positive component for moves of type I and II. For moves of type III and IV we choose the move which is most balanced in its components, i.e. where the $\Delta z(h)$ and $\Delta u(h)$ components are most equal. The ranking of different move type groups is done by always considering type III moves first, and type IV moves last, with move types I and II grouped together in the middle. To be able to properly rank move types I and II, their values need to be normalized.

We first define the ratio R by

$$R = \frac{\sum \text{abs}(z(h))}{w \sum \text{abs}(u(h))},$$

where the summations are taken over the currently available moves of type I

between the two measures making up the move evaluation function, with low values giving a bias towards selecting moves resulting in a lower integer infeasibility measure.

The normalized move values are

$$MoveType\,I : R1(h) = (\Delta z(h)/\Delta u(h))/R$$
$$MoveType\,II : R2(h) = (\Delta u(h)/\Delta z(h)) * R.$$

The preferred move selection from groups I and II is then

$$MoveType\,I\,and\,II : Max(R1(h), R2(h)).$$

Tabu status and aspiration criteria

We impose tabu restrictions on the variable that leaves the basis in a move. In other words, a variable that has just left the basis is not allowed to reenter the basis until its tabu tenure has expired.

Aspiration is by new global best. As mentioned, a new integer feasible solution may be encountered during a move evaluation without subsequently selecting the move leading to it. This is due to the process for evaluating and ranking the moves. Such a solution is, however, picked by the aspiration criterion, and will thus be selected.

For more detailed explanations of the workings of this section, the reader is again referred to Glover and Løkketangen (1997).