INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions

Edward Rothberg,

To cite this article:
Edward Rothberg, (2007) An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. INFORMS Journal on Computing 19(4):534-541. https://doi.org/10.1287/ijoc.1060.0189

INFORMS Journal on Computing
Vol. 19, No. 4, Fall 2007, pp. 534–541
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1904 | 0534

DOI 10.1287/ijoc.1060.0189
© 2007 INFORMS

An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions

Edward Rothberg
ILOG, Inc., Sunnyvale, California 94087, erothberg@ilog.com

Evolutionary algorithms adopt a natural-selection analogy, exploiting concepts such as population, combination, mutation, and selection to explore a diverse space of possible solutions to combinatorial optimization problems while, at the same time, retaining desirable properties from known solutions. This paper describes an evolutionary approach to improving solutions to mixed integer programming (MIP) models. We propose coarse-grained approaches to mutating and combining MIP solutions, both built within a large-neighborhood search framework. These techniques are then integrated within a MIP branch-and-bound framework. The resulting solution-polishing heuristic often finds significantly better feasible solutions to very difficult MIP models than do available alternatives. In contrast to most evolutionary algorithms, our polishing heuristic is domain-independent, requiring no structural information about the underlying combinatorial problem, above and beyond the information contained in the original MIP formulation.

Key words: mixed integer programming; metaheuristics; evolutionary algorithms
History: Accepted by William J. Cook, Area Editor for Design and Analysis of Algorithms; received September 2005; revised January 2006, March 2006; accepted May 2006. Published online in *Articles in Advance* July 25, 2007.

1. Introduction

Mixed integer programming (MIP) is a versatile, widely used technique for solving a variety of practical optimization problems. While MIP models of enormous size can often be solved to proven optimality, the current state of the art in solution techniques can leave significant optimality gaps for many important models. Fortunately, solutions that are not provably optimal can still provide significant practical value. It is therefore often important to find the best possible solution in a given amount of time.

Recent MIP heuristics have made significant progress toward that goal. Methods such as RINS and guided dives (Danna et al. 2005), and local branching (Fischetti and Lodi 2002) find high-quality solutions to difficult MIP models by exploring neighborhoods of a given feasible solution. The initial feasible solution can come from a variety of sources, including the MIP branch-and-bound search, a MIP heuristic (Bixby et al. 2000, Fischetti et al. 2005), a previous invocation of the same heuristic, or a problem-specific heuristic (Beck and Refalo 2002, Chabrier et al. 2003).

Given the success of these limited-neighborhood-exploration strategies, a natural question is whether more extensive neighborhood exploration might be productive. The combinatorial optimization literature is rich with neighborhood search frameworks, including tabu search (Glover and Laguna 1997), genetic algorithms (Mitchell 1996), simulated annealing (van Laarhoven and Aarts 1987), evolutionary algorithms (Holland 1975), and large neighborhood search (LNS) (Shaw 1998). These metaheuristics have been combined with exact methods to attack a variety combinatorial optimization problems (Puchinger and Raidl 2005). This paper proposes and evaluates a particular hybridization tailored to solving general MIP models. Specifically, we propose an evolutionary algorithm where the associated combination and mutation steps are performed using large neighborhood search. This algorithm is embedded inside the MIP branch-and-bound search. We find that our approach often produces solutions that alternative approaches are unable to find.

2. An Evolutionary Algorithm for MIP

Evolutionary algorithms typically maintain a population of solutions. In each phase of the algorithm, a new generation of solutions is created from the previous generation. The new generation is formed in a series of three steps:

- Selection: Pairs of candidate solutions are chosen, typically based on a fitness metric. The intent is that the fittest candidates produce the most descendants.
- Combination: Chosen pairs of solutions are combined (in a way that is meaningful within the problem domain) to produce offspring. The fitness of the offspring may be better or worse than that of its parents.
- Mutation: Random changes are introduced into some subset of the offspring (again in a way that is meaningful within the problem domain). The intent of this step is to increase diversity in the population.

Genetic algorithms (GAs) are a specific class of evolutionary algorithm that carries the biological analogy one step further. In a GA, solutions are described using sequences of bits (genes). Mutation is achieved by complementing one or more bits in the bit sequence. Combination (or crossover) is achieved by concatenating subsequences from the parent genomes. While it is possible to cast a MIP heuristic in such terms (Nieminen et al. 2003), it is quite difficult to pass desirable solution properties from parents to offspring with this representation. In particular, it is difficult to retain linear feasibility in the offspring. Our algorithm applies the biological analogy at a much higher abstraction level. We therefore refer to our approach using the more generic *evolutionary algorithm* label.

We now describe our evolutionary algorithm in terms of the crucial ingredients mentioned previously. We begin by discussing how the evolutionary heuristic is integrated into the MIP solution process. We then discuss the specific strategies we use for combination, mutation, and selection within the heuristic.

2.1. General Structure

A solution-improvement strategy obviously cannot exist in isolation. It must be seeded with one or more feasible solutions. We must therefore consider how the evolutionary algorithm will interact with a standard MIP solver.

A MIP model is generally solved by exploring a tree of continuous relaxations within a branch-and-bound framework. At each node in the search tree, the solver creates a pair of disjoint child relaxations by adding mutually exclusive bounds to the child nodes for a chosen branching variable (e.g., $x_j \leq 0$ or $x_j \geq 1$). Subtree exploration can be terminated if the relaxation at the root of the subtree is integer feasible, linearly infeasible, or has an objective value that is worse than that of the best known integer feasible solution. The solver can optionally perform several other operations at each node, including (i) generating cutting planes to tighten the continuous relaxation, (ii) performing node presolve to exploit logical reductions that are implied by previous branching decisions but are not captured in the linear relaxation, and (iii) node heuristics to try to generate feasible solutions from the current relaxation solution. Further details on how this is done within CPLEX can be found in Bixby et al. (2000).

When considering the question of how to integrate an evolutionary heuristic into this framework, one possibility would be to create an entirely disconnected process that would accept a solution from a (truncated) MIP tree search as input and produce an improved solution as output. We have chosen instead to perform a much tighter integration. In particular, we have chosen to allow the MIP solver to invoke the evolutionary algorithm at any node in the search tree. The MIP solver treats the evolutionary algorithm as a node heuristic, integrating newly found solutions into the MIP search state as it would any other heuristically generated solutions. Similarly, the MIP solver places solutions it finds during the tree search into the pool of solutions used by the evolutionary algorithm.

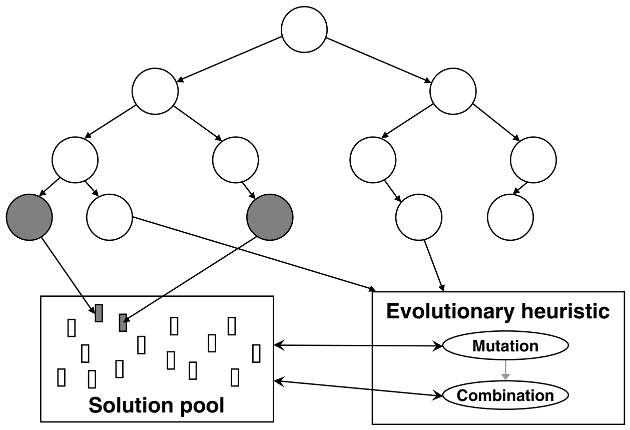

Figure 1 Integration of Polishing Algorithm Within the MIP Search Tree

Our evolutionary heuristic executes a fixed number of combine/mutate passes at each node of the MIP tree search. Repeated phases of the evolutionary algorithm are obtained by simply invoking the algorithm at regular intervals in the tree (e.g., every 100 nodes). This structure provides significant flexibility. For example, by invoking the evolutionary algorithm infrequently, and thus allowing the tree search itself to perform significant work, our polishing algorithm can work in tight cooperation with the tree search. Alternatively, our polishing algorithm can work in near isolation if it is invoked at every node. Integrating the heuristic with the search in this way also allows information to be shared between the tree search and the evolutionary algorithm (e.g., past solutions, reduced cost fixings, evolving lower bound information, etc.).

The overall structure of our polishing algorithm is illustrated in Figure 1. The shaded nodes represent MIP tree nodes where new integer feasible solutions are discovered and placed in the solution pool. Each invocation of the evolutionary heuristic performs one mutation phase and one combination phase, placing solutions it finds into the solution pool.

2.2. Population

Our evolutionary algorithm maintains a fixed-size pool of the P best distinct solutions found so far. As mentioned earlier, the pool contains solutions found during the standard MIP tree search, as well as solutions found by the evolutionary algorithm itself. The pool persists for the entire MIP tree search, so a solution found in an earlier invocation of the evolutionary algorithm is still available on the next invocation (provided it hasn't been pushed out by a better solution in the interim).

Since our evolutionary algorithm is a solution-improvement method, it can be invoked only when the pool contains a solution. While the combination step in an evolutionary algorithm requires two solutions, our polishing algorithm can augment the solution pool through its mutation process, as will be discussed shortly. It can therefore be applied even when the solution pool contains only a single solution.

2.3. Combination

In a typical evolutionary algorithm, combining a pair of solutions to form an offspring is an inexpensive, mechanical procedure. In a genetic algorithm, for example, combination is typically done using a crossover procedure that lines up the two solutions (expressed as bit vectors), chooses a crossover point, and then replaces the bit vector beyond that point in one parent by the corresponding portion in the other parent. In the MIP context, where respecting the linear constraints is often a challenging problem itself, it is unlikely that a similar mechanical procedure would be effective in general. Something more involved seems necessary in order to avoid losing the valuable information contained in the parent solutions.

As mentioned earlier, our combination algorithm uses a large-neighborhood-search approach (Shaw 1998) to combine pairs of solutions. In large-neighborhood search, a subset of the decision variables in a problem are fixed at their values in a known solution. The remaining problem is then solved, typically using the same method that would be used to solve the original problem. By shrinking the size of the problem, LNS exploits the combinatorial implosion that often occurs when problems sizes are reduced in difficult combinatorial problems. It is not unusual for a small number of fixings to decrease the difficulty of the remaining problem radically. Since LNS is a heuristic procedure, the solution of the sub-problem may also be truncated to limit the cost.

Our combination procedure is an extension of relaxation-induced neighborhood search (RINS) (Danna et al. 2005). In RINS, LNS fixings are determined by comparing the best known integer feasible solution and the relaxation solution at the current branch-and-bound node. Integer variables whose values agree in the two solutions are fixed. Values for other variables are determined by solving a MIP model on the unfixed variables.

Recall that our goal in the context of an evolutionary algorithm is to combine a pair of feasible solutions to form an offspring. The RINS notion can be extended to this context by now fixing variables whose values agree in both feasible solutions, again allowing the values for the other variables to be determined by solving a MIP. To limit the cost of the sub-MIP exploration, we truncate the tree-search process using an arbitrary node limit L. The best solution found in L nodes is added to the solution pool (unless it already appears there, or unless the pool is full and the objective of the new solution is no better than the objective value of the worst solution in the pool). An example of the combination process is illustrated in Figure 2. Question marks in the offspring solution vector represent values that the MIP solver is asked to compute.

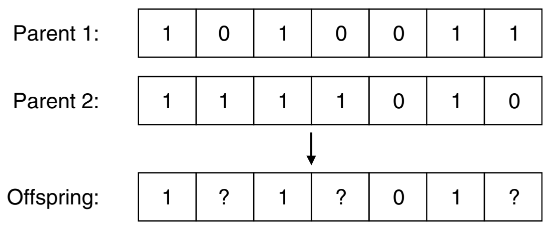

Figure 2 **Combination Process**

While an evolutionary algorithm typically combines pairs of solutions, we have found it advantageous to perform combinations among larger sets as well. The extension of our combination method from two solutions to many is quite simple—variables are fixed only when their values agree in all solutions. The resulting sub-MIP model is larger and more difficult than it would be when just two solutions are combined, but the MIP solver can sometimes find solutions in this less-constrained problem that it does not find otherwise.

A fixed number C of combination steps are performed in each pass of our evolutionary algorithm. The details of how parent solutions are selected will be discussed shortly.

2.4. Mutation

Without some sort of mutation strategy, the solution pool quickly becomes quite homogeneous. This lack of diversity is compounded in each subsequent generation. A typical mutation strategy will change a small piece of the solution (often a single bit in a genetic algorithm), and add the resulting solution back into the solution pool. In the MIP context, we are again constrained by the fact that it can be extremely difficult to identify a small change that retains linear feasibility. We again rely on LNS to explore a set of alternative solutions.

We build a mutated solution by first choosing a random *seed* solution from the solution pool. This solution is then used to choose variable fixings in the

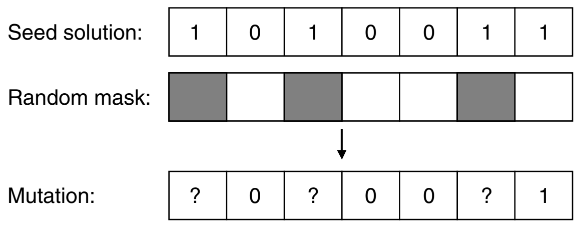

Figure 3 **Mutation Process**

original MIP model. Specifically, we fix a random subset of the integer variables in the model to the values they take in the seed solution. An example of our mutation process is shown in Figure 3. We then solve the resulting sub-MIP problem, again truncating the tree exploration if necessary. The resulting solution is considered for addition to the solution pool.

A crucial issue in this mutation strategy is the choice of the number of variables to fix. If too many variables are fixed, then the sub-MIP will not explore a very large space of possible alternatives. If too few are fixed, the space of possible alternatives may be too large, and the tree search may not be significantly easier than the search on the original model. The best fixing strategy is highly problem-dependent, and can often depend on the specific seed solution within a single problem. We have therefore developed an adaptive approach. An initial fraction f is chosen arbitrarily (e.g., 50%). The associated fixings are performed, and L nodes of the sub-MIP are explored. If the sub-MIP solution is worse than the seed solution, or if no solution is found, then we assume that the sub-MIP is too difficult and we increase the target fixing fraction. If the sub-MIP finds the seed solution (which is always a feasible solution to the sub-MIP), but is unable to find a better solution, then we assume that the sub-MIP is too constrained and we reduce the target fixing fraction. If an improved solution is found, the target fixing fraction is left unchanged. We have found that this approach settles on a value for the fixing fraction quickly, and the fraction subsequently remains in a narrow range. Tests with several models and a range of specific choices for the fixing fraction have shown that the dynamic scheme settles on a fixing fraction that is typically quite close to the best choice. Like the solution pool itself, the value of the fixing fraction f persists from one invocation of the evolutionary algorithm to the next.

While it is most common in evolutionary algorithms to perform mutation after combination, recall that in our context we may begin the evolutionary algorithm with few or even just one feasible solution. We therefore perform mutation first (on M randomly chosen seed solutions from the solution pool), hopefully expanding the solution pool for the subsequent combination phase.

2.5. Selection

Another important issue in an evolutionary algorithm is the selection of parents for the combination process. The selection process should bias the choice towards parents with favorable objective values, but the bias should not be too strong, since much of the strength of an evolutionary algorithm comes from its exploitation of diversity in the solution pool. We experimented with a variety of different selection procedures, but found that the simplest worked as well as any others we tried. Given the current solution pool, sorted in order of worsening objective value (with increasing age as a secondary key), our selection procedure chooses a random solution from the solution pool as the first parent, and then chooses the second parent randomly from amongst those that come before the first in the sorted list. This introduces a modest bias towards solutions with better objective values.

We mentioned earlier that we also consider combinations among more than two parents. While the selection approach described above could be extended to larger sets, we instead chose to perform combination only on sets containing exactly two parents or sets containing all solutions in the pool. The latter case, of course, requires no selection step.

Selection is also relevant for mutation. To select a mutation seed, we choose a pair of solutions as in the selection process for combinations, but we only use the solution with the better objective as the mutation seed.

As mentioned earlier, one invocation of our polishing algorithm performs a fixed number C of combinations. In contrast to many evolutionary algorithms, new solutions found during the combination/mutation process in our algorithm are added to a unique solution pool immediately. Our algorithm therefore has no notion of building a new generation of solutions from the previous generation; pairs of candidates from different generations can be combined. We have found that this approach gives a more effective procedure in our context. Our combination and mutation steps are quite expensive, so only a limited number can be performed. It is therefore important to allow new, improved solutions to participate as quickly as possible.

2.6. Putting It Together

To make the overall process hopefully more clear, we now lay out the steps performed in one invocation of our polishing algorithm (i.e., the steps inside the bottom right box in Figure 1):

(1) Mutation—repeat M times

(a) Choose a seed solution from the solution pool.

(b) Fix a fraction f of variables to their values in the chosen solution.

(c) Explore L nodes of the MIP search tree for the resulting sub-MIP.

(d) Add the best solution found to the solution pool.

(e) Adjust f, based on the objective value of the solution found.

(2) Combination—repeat C times

(a) Choose a pair of solutions from the solution pool (occasionally choose all solutions).

(b) Fix variables whose values agree in all of the chosen solutions.

(c) Explore L nodes of the MIP search tree for the resulting sub-MIP.

(d) Add the best solution found to the solution pool.

As mentioned earlier, passes of this algorithm are interleaved with node exploration in the MIP tree.

3. Benchmarks

3.1. Benchmark Selection

Evaluating the effectiveness of a solution-improvement strategy can be quite difficult. To understand the potential benefit, we must identify benchmark models for which it is extremely difficult to find good solutions, and we must also decide when to invoke the improvement strategy. Attempting to improve a low-quality solution may lead to the conclusion that the method is quite effective, but will say little about whether it is more effective than the alternatives. Attempting to improve an optimal solution, on the other hand, could potentially lead to the conclusion that the method is ineffective. We therefore need to be careful to choose initial feasible solutions that are good, but not too good. We also need to set the results in the right context by comparing against alternative approaches.

In light of these issues, we have chosen a set of test models and seed solutions according to the following criteria. We have chosen models for which finding good solutions is known to be quite difficult. For each model, we provide significant opportunities for the branch-and-cut search to explore the search tree in order to find good feasible solutions. We also allow the best available improvement heuristics (RINS, guided dives, and local branching) to improve on these solutions. Specifically, we explore 50,000 nodes of the branch-and-cut tree for each model, using guided dives to guide the tree search, performing RINS once every 100 nodes in the tree, and performing local branching on new solutions found by either the branch-and-cut search or the RINS search. This gives RINS 500 opportunities to find an improved solution, and it ensures that local branching is given the opportunity to improve the final solution. In light of past experience with these heuristics, it is reasonable to assume that further improvements in the resulting solutions will not come easily.

To avoid penalizing our method for not improving optimal solutions, we discard models where the best solution found after 50,000 nodes is also the best known solution. Note that for some models, the only method that finds a best known solution is our polishing algorithm. This criterion introduces some bias toward models where the polishing algorithm is particularly effective. However, excluding such models would introduce the opposite bias. We have chosen simply to note that these dueling biases exist.

Once the initial 50,000-node tree exploration is complete, we allow three alternative approaches to continue the search for better solutions. We have (arbitrarily) chosen to allow each to proceed for half the time taken by the initial 50,000-node tree exploration. The first alternative is to allow default branch-and-cut search to explore the MIP search tree. The second is to continue with RINS, guided dives, and local branching. The final alternative is our polishing heuristic, invoked at every node in the branch-and-cut tree. As mentioned earlier, we could have chosen to invoke the method less frequently in order to obtain a blend of the various alternatives. We chose not to explore the continuum between the various methods.

3.2. Benchmark Machine and Software

All our tests are performed using CPLEX 9.1. Tests that rely on RINS, guided dives, and local branching use the implementations from the released product. We built a custom implementation of our solution-polishing heuristic for the purposes of performing these tests. Our tests were performed on an AMD Opteron 244 processor with a clock speed of 1.8 GHz, a 1 MB secondary cache, and 6 GBytes of main memory.

Our polishing algorithm requires us to choose specific values for several constants. Based on significant experimentation, we have chosen the following values for the tests that follow. Our truncated MIP search always explores at most $L = 500$ nodes. Our solution pool contains $P = 40$ solutions. We generate $M = 20$ mutation candidates, and perform $C = 40$ combination steps. Of these 40 combination steps, 39 combine pairs of solutions, while one combines all solutions in the pool. Our initial fixing fraction is $f = 0.5$. Our initial increment for the fixing fraction is $i = 0.2$. We reduce i by 25% after every invocation of our solution-polishing method, until it hits 0.01, at which point it is no longer decremented. The fixing fraction f is incremented or decremented by the then-current value of the increment i, depending on

Table 1 Benchmark Model Statistics

Instance	Reference	*n*	*b*	*i*	*m*	Time	Best obj.
glass4	Martin et al. (2003)	322	302	0	396	131	1,200,012,600
liu	Martin et al. (2003)	1,156	1,089	0	2,178	1,766	1,108
mkc	Martin et al. (2003)	5,325	5,323	0	3,411	2,317	−563.846
protfold	Martin et al. (2003)	1,835	1,835	0	2,112	65,096	−31
sp97ar	Martin et al. (2003)	14,101	14,101	0	1,761	6,861	661,716,162
swath	Martin et al. (2003)	6,805	6,724	0	884	1,310	467.4075
timtab2	Martin et al. (2003)	675	113	181	294	2,343	1,096,557
bg512142	Surie (2002)	792	240	0	1,307	5,182	185,824
dg012142	Surie (2002)	2,080	640	0	6,310	13,886	2,317,029.5
B2C1S1	Pochet and Van Vyve (2004)	3,872	288	0	3,904	28,019	25,687.91
pharma1	ILOG (2002)	125,840	1,320	5,517	71,178	25,854	21,458,751,679
sp97ic	van Hoesel et al. (2001)	12,497	12,497	0	1,033	1,827	428,580,586
sp98ar	van Hoesel et al. (2001)	15,085	15,085	0	1,435	2,548	529,814,785
sp98ic	van Hoesel et al. (2001)	10,894	10,894	0	825	1,567	449,144,758
UMTS	van Hoesel et al. (2001)	2,947	2,802	72	4,465	1,573	30,091,886
rococoB10-011001	Chabrier et al. (2003)	4,456	4,320	136	1,677	6,262	21,223
rococoB11-110001	Chabrier et al. (2003)	12,431	12,265	166	8,148	49,408	42,444
rococoB12-111111	Chabrier et al. (2003)	9,109	8,778	331	8,978	27,500	37,167
rococoC10-100001	Chabrier et al. (2003)	5,864	5,740	124	7,596	32,034	16,664
rococoC11-010100	Chabrier et al. (2003)	12,321	12,155	166	4,010	140,858	20,889
rococoC12-100000	Chabrier et al. (2003)	17,299	17,112	187	21,550	79,601	35,512
rococoC12-111100	Chabrier et al. (2003)	8,619	8,432	187	10,842	20,628	35,909
ljb2	Beck and Refalo (2002)	771	681	0	1,482	480	0.507679
ljb7	Beck and Refalo (2002)	4,163	3,920	0	8,133	3,055	0.099038
ljb9	Beck and Refalo (2002)	4,721	4,460	0	9,231	5,166	0.739
ljb10	Beck and Refalo (2002)	5,496	5,196	0	10,742	7,981	0.436284
ljb12	Beck and Refalo (2002)	4,913	4,633	0	9,596	8,399	0.381528

the results of the sub-MIP search. The sub-MIPs are solved using default CPLEX settings, with the exception of the MIP node limit, which is set to *L*.

3.3. Benchmark Models

Table 1 lists the models we have chosen for our benchmark tests, and provides various relevant statistics about these models. The first column gives the name of the model, and the second gives the source. The next five columns provide size information for each model, including the number of variables (*n*), the number of those that are binary (*b*) and general integer (*i*), and the number of constraints (*m*). The next column shows the time taken to explore 50,000 nodes of the CPLEX branch-and-cut tree on our benchmark system using CPLEX 9.1 with our baseline settings (using guided dives to guide the tree search, invoking RINS at every 100 nodes in the tree, and applying local branching on the last solution found at each node). The solutions computed by this run are used to seed the solution pool for our polishing strategy.

The last column shows the objective value for the best solution known to us for each model. These solutions come from several sources. Several come from problem-specific methods (described in the cited papers). Others come from extremely long runs of CPLEX.

4. Computational Results

Table 2 shows the results of our computational tests. The second column provides information about the objective value found after 50,000 nodes for each model. Specifically, this column shows the gap between the objective value for the solution found after 50,000 nodes and the best known objective value (computed as the absolute value of the difference between the two objective values, divided by the absolute value of the best known objective value). The next three columns show the same ratios for the three alternatives we consider for further improving the solution obtained after 50,000 nodes: (i) default CPLEX settings, (ii) a continuation of RINS, guided dives, and local branching, and (iii) our solution polishing method. The best solution obtained for each model is shown in bold. The table shows that our polishing method is consistently more effective than the two alternatives.

One point to note from the table is that our polishing heuristic is not the most effective approach for the ROCOCO model set. We have found that for these models, our dynamic scheme consistently chooses a fixing fraction of only 5%–10%. The resulting sub-MIP models are quite large and quite difficult, and generally don't experience the combinatorial implosion that LNS achieves for other models. However, further

Table 2 Computational Results—Relative Gap Between Solution Found by Method and Best Known Solution

	Relative solution quality (versus best known)			
		After 50% additional time		
Instance	Initial 50K nodes GD + LB + RINS	Defaults	GD + LB + RINS	Polishing
glass4	0.34722	**0.34722**	**0.34722**	**0.34722**
liu	0.09747	0.09747	0.09567	**0.03430**
mkc	0.00020	0.00020	0.00020	**0.00000**
protfold	0.12903	0.12903	0.12903	**0.06452**
sp97ar	0.00090	0.00090	0.00081	**0.00056**
swath	0.02517	0.02517	0.02517	**0.02272**
timtab2	0.07545	0.07545	0.07545	**0.06772**
bg512142	0.04287	0.04287	0.04287	**0.00000**
dg012142	0.26215	0.26215	0.26198	**0.26137**
B2C1S1	0.00707	0.00707	0.00707	**0.00093**
pharma1	0.00288	0.00288	0.00288	**0.00129**
sp97ic	0.00360	0.00360	0.00360	**0.00000**
sp98ar	0.00083	0.00083	0.00083	**0.00079**
sp98ic	0.00289	0.00289	**0.00018**	0.00234
UMTS	0.00107	0.00107	0.00107	**0.00106**
rococoB10-011001	0.02917	0.02328	**0.00820**	0.01965
rococoB11-110001	0.03058	0.03058	**0.02938**	**0.02938**
rococoB12-111111	0.02919	0.02919	**0.00000**	0.01369
rococoC10-100001	0.06025	0.06025	**0.05911**	0.06025
rococoC11-010100	0.04050	0.01249	**0.00326**	0.04050
rococoC12-100000	0.01349	**0.01349**	**0.01349**	**0.01349**
rococoC12-111100	0.01033	0.01033	0.01033	**0.00994**
ljb2	0.01574	0.01574	0.00435	**0.00000**
ljb7	0.24904	0.24904	0.24747	**0.17195**
ljb9	0.77430	0.53763	0.57458	**0.32891**
ljb10	0.03254	0.03254	0.03254	**0.03196**
ljb12	0.32932	0.32932	0.25586	**0.18556**

experimentation revealed that larger fixing fractions did not produce better results, so the problem is not simply a failure of the adaptive scheme. This issue will require further investigation.

5. Discussion

Evolutionary algorithms rely heavily on diversity in the solution pool for their success. Unfortunately, most members of the solution pool in our approach are themselves found by neighborhood search methods—by RINS, guided dives, local branching, or previous passes of the solution-polishing heuristic. Diversity is therefore quite limited. We tried a few approaches to increasing diversity. One approach was to impose a diversity restriction on new members of the solution pool. Specifically, we would add a new member only if it differed from all other members in the values of at least k integer variables, for some chosen constant k. We also investigated an idea inspired by local branching (Fischetti and Lodi 2002), where we added a constraint to the original MIP that required the solution of the modified MIP to differ in at least k binary variables and computed solutions to that model. Unfortunately, neither approach produced an improvement in the behavior of the overall method.

As mentioned earlier, metaheuristics and branch and bound have been combined in several other contexts. The survey by Puchinger and Raidl (2005) proposes a taxonomy for describing such hybrid methods. Within this taxonomy, our approach would be considered an *integrative combination*, meaning that we embed one approach within the other. Our approach can actually be viewed as a recursive integrative combination, since we embed an evolutionary algorithm within the MIP branch and bound, while simultaneously embedding MIP branch and bound within the evolutionary algorithm.

Regarding the specifics of our hybrid method, we note that LNS has been used in a MIP context before. In particular, it has been used for problems from job-shop scheduling (Applegate and Cook 1991) and crew scheduling (Anbil et al. 1991). One difference with our approach is that previous work has relied on domain information to identify sets of decision variables to fix. Since our approach was designed to be applicable to arbitrary MIP models, it required a more general scheme for choosing the set of variables to fix. We also placed this LNS search within an evolutionary framework to exploit diversity in a solution pool.

Our solution-combination method can be viewed as a generalization of an approach used by Applegate et al. (1998) to obtain high-quality solutions to traveling-salesman problems. Their approach forms the union of the set of edges in a collection of high-quality TSP tours, and then solves the TSP problem on the resulting graph exactly. By way of analogy, edges that appear in one tour but not another can be viewed as variables that take different values in different solutions.

Another piece of related work is a hybrid MIP/metaheuristic approach by French et al. (2001). Their approach also shares information between a branch-and-bound search and a metaheuristic, but their metaheuristic (a genetic algorithm) operates within the context of a specific problem class, MAX-SAT. Since any bit vector is feasible for the MAX-SAT problem, issues of achieving or maintaining feasibility in the GA candidate pool did not arise. While their approach could be extended to other problem domains, they do not consider mutation or combination strategies for general MIP.

Nieminen et al. (2003) have proposed a GA for finding feasible solutions to MIP models. As mentioned earlier, their method is a genetic algorithm; mutation involves flipping a bit in the solution representation, and crossover happens through the concatenation of subsequences from two parents. Their approach employs a repair procedure, based on limited backtrack, to reduce linear infeasibility, but little emphasis is placed on transferring parent feasibility to child solutions when performing crossover. Their primary emphasis is on finding a first feasible solution.

6. Conclusions

This paper has shown that neighborhoods of good MIP solutions continue to be fertile ground when mining for better solutions. By employing the natural-selection analogy of evolutionary algorithms, combined with large-neighborhood search to explore possible mutations and combinations, we were able to find significantly better solutions than the best existing heuristics for several very difficult MIP models.

Acknowledgments

The author thanks Renaud Dumeur and Paul Shaw for helpful comments and discussions. The author also thanks the anonymous referees for their suggestions for improving this paper.

References

Anbil, R., E. Gelman, B. Patty, R. Tanga. 1991. Recent advances in crew-pairing optimization at Americal Airlines. *Interfaces* **21** 62–74.

Applegate, D., W. Cook. 1991. A computational study of the job shop scheduling problem. *ORSA J. Comput.* **3** 149–156.

Applegate, D., R. Bixby, V. Chvátal, W. Cook. 1998. On the solution of traveling salesman problems. *Documenta Mathematica, Extra Volume Proceedings ICM III* (*1998*), 645–656, http://citeseer.ist.psu.edu/article/applegate98solution.html.

Beck, J. C., P. Refalo. 2002. Combining local search and linear programming to solve earliness/tardiness scheduling problems. *Proc. Fourth Internat. Workshop Integration AI and OR Techniques Constraint Programming Combin. Optim. Problems* (*CP-AI-OR'02*)*, LeCroisic, France*, 221–235.

Bixby, R., M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling. 2000. MIP: Theory and practice—Closing the gap. *System Modelling and Optimization: Methods, Theory, and Applications*. Kluwer Academic Publishers, Boston, MA, 19–49.

Chabrier, A., E. Danna, C. Le Pape, L. Perron. 2003. Solving a network design problem. *Ann. Oper. Res.* **130** 217–239.

Danna, E., E. Rothberg, C. Le Pape. 2005. Exploring relaxation induced neighborhoods to improve MIP solutions. *Math. Programming* **102** 71–90.

Fischetti, M., A. Lodi. 2002. Local branching. *Math. Programming* **98** 23–47.

Fischetti, M., F. Glover, A. Lodi. 2005. The feasibility pump. *Math. Programming* **104** 91–104.

French, A., A. Robinson, J. Wilson. 2001. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *J. Heuristics* **7** 551–564.

Glover, F., M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Boston, MA.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

ILOG. 2002. *CPLEX Customer Model*. ILOG, Mountain View, CA.

Martin, A., T. Achterberg, T. Koch. 2003. MIPLIB 2003, http://miplib.zib.de.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA.

Nieminen, K., S. Ruuth, I. Maros. 2003. Genetic algorithm for finding a good first integer solution for MILP. Technical Report 2003/4, Imperial College, London, UK.

Pochet, Y., M. Van Vyve. 2004. A general heuristic for production planning problems. *INFORMS J. Comput.* **16**(3) 316–327.

Puchinger, J., G. R. Raidl. 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. J. Mira, J. R. Álvarez, eds. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. Springer, Berlin, Germany, 41–53.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *Fourth Internat. Conf. on Principles and Practice of Constraint Programming, Pisa, Italy*, 417–431.

Surie, C. 2002. MLCLSP test instances. http://www.bwl.tu-darmstadt.de/bwl1/forschung/ti_mlclsp/ti_mlclsp.php?FG=bwl1.

van Hoesel, S., J. W. Goessens, L. Kroon. 2001. A branch-and-cut approach to line planning problems. Technical Report, Erasmus University, Rotterdam, The Netherlands.

van Laarhoven, P. J. M., E. H. L. Aarts. 1987. *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Boston, MA.