A pre-processing aware RINS based MIP heuristic

Thiago M. Gomes[1], Haroldo G. Santos[1], and Marcone J. F. Souza[1]

Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
Ouro Preto – MG – Brasil

Abstract. This paper proposes an adaptation of the RINS MIP heuristic which explicitly explores pre-processing techniques. The method systematically searches for the ideal number of fixations to produce subproblems of controlled size. These problems are explored in a Variable Neighborhood Descent fashion until a stopping criterion is met. Preliminary experiments implemented upon the open source MIP solver COIN-OR CBC are presented.

1 Introduction

One of the most important techniques for solving complex optimization problems is Mixed Integer Programming (MIP). A MIP problem involves a set of variables, a set of constraints on these variables, a set of integrality constraints and a linear objective function to optimize.

MIPs are typically solved by branch-and-bound or branch-and-cut techniques. These approaches explore a tree of relaxations of the original MIP, in which each node in the tree is divided into two disjointed sets by imposing limiting restrictions upon an integer variable. Even tough MIP solvers can be applied to a variety of problems, their performance in producing good feasible solutions and strong dual bounds greatly differs for different applications and their respective formulations. Thus, operations research practitioners often consider the use of specifically tailored heuristics such as Tabu Search [1], Genetic Algorithms [2], [3], Reconnect paths [4], among others, which objective is only in the production and improvement of feasible solutions. In recent years, the application of MIP solvers in different domains has motivated the development of many MIP heuristics, such as Feasibility Pump [12][13], Local Branching [14] and RINS[11].

Among the solvers which have their source code available, we can highlight the COIN-OR CBC [16] [17]. It is a solver of linear and integer programs. The package includes features such as pre-processing, cutting planes, heuristics and branching strategies. Although it was initially designed to be used as a library, it includes an independent solver callable by the command line. It is possible to use the file formats .lp and .mps. It can also run in parallel to take advantage of multi-core computers.

In this paper we propose modifications in MIP heuristic Relaxation Induced Neighborhood Search (RINS) proposed by [11] in the literature. The idea is to

search for an ideal number of variables to be fixed in a specific problem. If this number is too small, the search space may be too large to be searched efficienly. On the other hand, a large number of fixations may restrict the search space so much that no improved solution will be found. All our implementations were coded using the COIN-OR CBC libraries.

The remainder of this paper is divided as follows. Section 2 presents the literature review considering integer programming problems and metaheuristics. Section 3 describes the proposed adaptive heuristics RINS(pRins) and the methodology used. In section 4, the test instances are described, while in Section 5 the experiments and computational results are presented. The last section concludes the paper and suggests possible improvements.

2 Literature Review

The heuristics were originated in Operations Research and Artificial Intelligence communities. At first, combinations of heuristics were not explored, since they worked well separately. The motivation for hybridization emerged aiming at exploring the benefits of the synergy between the methods. However, it is not trivial to find good combinations [5].

[5] presents a survey on hybridization of metaheuristics with other optimization methods for solving problems of combinatorial nature. The authors emphasize the importance of hybrid methods combining features of diversification and intensification when searching for a solution. Among the methods of integer programming, we highlight those based on Lagrangian relaxation, as well as iterative heuristic (LPA, and IIRH IRH), which fix and solve the subproblems encountered at every stage.

For [6], the reason for using heuristics is that they must help MIP finding good solutions earlier, thus avoiding search in regions of low quality in the tree, and at the same time, they must enhance the search for promising regions. In his paper it is presented a 0-1 heuristic , stand-alone implementation, in other words, that is independent of branch-and-bound. It is built around a merit function by measuring the completeness of the solution. The method involves four steps: gradient-based pivoting, pivoting poll, cuts convexity/intersection and exploration of the tree of variables blocks.

While solving a problem in a Linear Programming based branch-and-bound algorithm, there are usually two solutions available, one that meets the requirements of completeness of variables, but is not optimal, and another one that is a fractional solution which doesn't meet the requirements of variables completeness, however has a better value. The Relaxation Induced Neighborhood Search(RINS) method, proposed by [11], is based upon this assumption and fixes the variables that are equal in both solutions, as they meet both criteria. Then a cut is added, based on the current value of the objective function and the subproblem is solved with the remaining variables. The method has characteristics which are similar to Path Relinking [4], since it is also connecting two solutions.

In [7], the authors apply two tree search techniques to the Traveling Salesman Problem: Local Branching (LB) and Sliced Neighborhood Search(SNS). While LB produces intensification in the region of the incumbent solution exploring nearby space, the SNS technique works diversifying the search. The SNS technique improves the incumbent solution by the random exploration of distant spaces in the neighborhood. It explores the space considering the disparities between the incumbent solution and the neighbors. At each iteration, a piece of the neighborhood is explored by choosing a set of variables or using short time limits. The combined result of the two techniques has proven itself to be satisfactory, finding better quality solutions.

[8] presents the heuristic Distance Induced Neighborhood Search (DINS). The idea of this heuristic is to use a metric distance between the linear relaxation solution and the current integer solution, exploring the nodes generated by the search. The DINS method incorporates hard-fixating and soft-fixating, rounding variables according to the metric defined. It follows the intuition that good solutions are close to the relaxed solution. The method also considers a criterion to avoid excessive fixations, in case many integer variables are fixed.

[15] describes an evolutionary approach to improve solutions to mixed integer programming (MIP) models. Evolutionary algorithms adopt a natural-selection analogy, exploring 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. The proposed method maintains a fixed-size pool of the P best distinct solutions found so far, to perform the operations of combination and mutation. For mutation, a solution is chosen, and then a percentage of variables are fixed. The resulting sub-MIP is solved, the best solution is added to the pool and the percentage of fixation is updated. The combination chooses a pair of solutions, fix variables whose values agree in all of the chosen solutions. Then the sub-MIP is solved and adds the best solution found to the solution pool. The experiments showed satisfactory results.

The heuristic Relaxation Enforced Neighborhood Search (RENS) presented in [9], works with large neighborhood search. The method builds a sub-problem considering the viable rounding of some fractional point - usually the optimal LP relaxation of the original MIP. The current solution is the starting point and neighborhoods are defined by using fixations and adding restrictions. The RENS idea is fixing variables with integer value in the relaxed solution and searching the remaining solutions, rounding to the nearest integer.

In [10] a beverage production plant is modeled using mixed-integer programming, and it involves lot sizing, decisions planning and dependent time machine preparation. It proposes a fixation and relaxation heuristic to explore the problem, since CPLEX does not satisfactorily resolve instances of the problem. In this procedure, the set of integer variables is divided into disjunctive sets. At each iteration, the variables of one of the sets are fixed, while the others are relaxed, and the resulting submodel is resolved. The sets were divided considering the production periods.

3 The pRINS MIP heuristic

The heuristic developed in this work is based on the method RINS. In this method, the following steps are taken at each node in the branch-and-cut tree:

1. fix the variables with the same value of the incumbent solution and the relaxed solution;
2. set a cut based on the value of the objective function in the current incumbent solution;
3. solve the sub-MIP with the remaining variables.

The RINS sets all the variables that are equal in both solutions. Depending on how many fractional variables appear on the fractional solution, the number of the fixes in the resulting neighborhood can be too small or too large. The proposed method explores existing pre-processing techniques to quickly generate sub-problems of controled size. These sub-problems are are solved in a VND(Variable Neighborhood Descend) [18].

Algorithm 1: pRINS

Input: $Sol_f, Sol_i, mip, Req_{size}, N_{atpmax}, Dif_p$
Output: Sol^*
$Sol^* \leftarrow Sol_i$;
X_p receives the sort of the variables according to their priority for fixing considering the integer and fractional solution (Sol_i, Sol_f);
repeat
  $fix_{size} \leftarrow$ buildSizes$(X_i, X_p, N_{atpmax}, Req_{size}, Dif_p, Lim_{Rel})$;
  $mip'' \leftarrow$ createProblem$(X_i, X_p, fix_{size}, mip)$;
  if *mip'' relaxation cost indicates possible improvement* **then**
    $Sol_{rins} \leftarrow$ solve mip'';
    if *found better solution* **then**
      $Sol^* \leftarrow Sol_{rins}$;
      Recalculate the vector of priorities (X_p) considering Sol^*, Sol_f ;
    else
      Increase the required size Req_{size};
    end
  else
    Increase the required size Req_{size};
  end
until *the required size is smaller than the total number of variables and time limit not exceeded* ;
return Sol^*;

The method pRINS(pre-processing aware RINS), algorithm 1, establishes a stopping criterion when applying this adapted heuristic. Given an integer solution (Sol_i), fractional solution (Sol_f), a required size to created problem (Req_{size}), maximum number of attempts for building a subproblem with the desired size(N_{atpmax}), acceptable percentage difference between desired and founded

size (Dif_p) and the original problem MIP (mip). Initially an ordered array indicating the priorities of variables for fixation is created. To define them, these vector variables are ordered according to the difference in magnitude between the value in the integer solution and fractional solution. The variables whose difference in (Sol^*) and (Sol_f) is zero are the first ones of this vector and the others are placed in this array in ascending order of difference. Then the method buildSizes, defined in algorithm 2, is used to determine the number of fixations (fix_{size}) necessary to achieve the required size problem. It is created a new problem(mip'') which is pre-processed, considering the number of fixations determined previously. If the relaxation cost does not indicates possible improvement, then the size of the problem increases. Otherwise, the problem is solved, if the best solution is found, the integer solution is updated, priorities recalculated, trying to solve the new problem. If the solution is worse, the size of the problem increases (Req_{size}) in percentage to consider a larger search space.

Algorithm 2: buildSizes

Input: $X_i, X_p, N_{atpmax}, N_{des}, Dif_p, Lim_{Rel}$
Output: N_{fix}
$N_a \leftarrow 0$;
$L_l \leftarrow 0$;
$L_u \leftarrow maxSize_{last}$;
$N_{fix} \leftarrow (L_l + L_u)/2$;
$Dif_{max} \leftarrow 0$;
repeat
  $N_a + +$;
  $mip'' \leftarrow$ createProblem(X_i, X_p, N_{fix}, mip);
  $Prob_{size} \leftarrow$ nvars(mip'');
  if $|N_{des} - Prob_{size}| \leq N_{des} * Dif_p$ **then**
    return N_{fix};
  end
  if $Prob_{size} == 0$ **or** $Prob_{size} < N_{des}$ **or** $Lim_{Rel} \geq 0.99 * cost(S_i)$ **then**
    $L_u \leftarrow N_{fix}$;
    $L_l \leftarrow N_{fix}/2$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  else
    $L_l \leftarrow N_{fix}$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  end
until $N_a < N_{atpmax}$ **or** $L_l < L_u$;
return N_{fix};

The method buildSizes, is used to determine the ideal number of fixations. Given the desired size of the problem (N_{des}), which in the beginning is the size of a mip that is small enough to be quickly solved, the total number of variables (size(X_i)), maximum number of retries (N_{atpmax}), the vector of priorities fixation (X_p), vector of integer variables (X_i), the MIPs are created and the

number of fixations is found. To reach the desired size, the number of fixations is changed according to a binary search until the number of attempts is exhausted, the binary search ends or size is achieved. Initially, the upper limit of fixation is the total number of variables in the problem. This upper limit is changed at the end of each binary search, being upgraded to the number of fixations used in the last sub-problem solved. This control is made through the variable ($maxSize_{last}$). This method calls another one (createProblem, defined in algorithm 3) which checks the feasibility of the resulting subproblem. After the problem is built, nvars function returns the number of free variables of the problem. The search ends when the number of free variables($Prob_{size}$) approaches the desired size(N_{des}), taking a percentage of tolerance(Dif_p). Otherwise, the search terminates if the maximum number of attempts(N_{atpmax}) is reached, or the upper limit(L_u) is less than or equal to lower limit(L_l).

Algorithm 3: createProblem

Input: X_i, X_p, N_{fix}, mip
Output: $mip^{'}$
$mip' \leftarrow fix(mip, N_{fix}, X_p, X_i)$;
$mip'' \leftarrow preprocessed(mip')$;
if $mip''notfeasible$ **then**
  $mip'' \leftarrow$ null ;
end
return mip'';

The method createProblem, in algorithm 3, will build mip problems, considering a number of variables to be fixed (N_{fix}), the vector of priorities (X_p). The function $fix(mip, N_{fix}, X_p, X_i)$ will create a new problem, from the fixing of (N_{fix}) first variables of vector (X_p), through the definition of upper and lower limits of each one by the value in X_i. Next, the created problem is preprocessed, generating another (mip''). If the mip (mip'') is viable, the method returns the size of the new problem. However if unfeasible, returns 0 as the size.

4 Characterization of Instances

The models (instances) to be used in this work are all related to binary problems. They were obtained from two groups:

- 25 MIPLIB library problems http://miplib.zib.de/ [20].
- 12 nurse scheduling problems used in the International Nurse Rostering Competition 2010 [19]

A detailed description of the instances, containing the number of binary variables, the number of constraints and number of non-zero values in constraints, is available in table 1.

We can see the diversity of binary problems that will be used, with this number of constraints and variables quite different.

Table 1. Nurse Scheduling and MIPLIB Library Instances

instance	binary variables	constraints	non-zero
long01	51,695	17,241	1,011,556
long-hidden01	61,950	28,370	1,064,380
long-hint01	61,550	27,480	1,061,430
long-late01	61,750	27,875	1,062,795
medium01	29,605	8,668	621,829
medium-hidden01	36,690	16,070	635,220
medium-hint01	34,050	14,062	622,800
medium-late01	34,050	14,062	622,800
sprint01	3,522	10,230	204,000
sprint-hidden01	10,308	3,332	202,420
sprint-hint01	11,630	5,032	208,410
sprint-late01	11,630	5,032	208,410
air04	8,904	823	72,965
bley-xl1	5,831	175,620	869,391
cov1075	120	637	14,280
eil33-2	4,516	32	44,243
eilB101	2,818	100	24,120
iis-100-0-cov	100	3,831	22,986
iis-bupa-cov	345	4,803	38,392
iss-pima-cov	768	7,201	71,941
macrophage	2,260	3,164	9,492
mine-166-5	830	8,429	19,412
mine-90-10	900	6,270	15,407
n3div36	22,120	4,484	340,740
n3seq24	119,856	6,044	3,232,340
neos-1109824	1,520	28,979	89,528
neos-1337307	2,840	5,687	30,799
neos18	3,312	11,402	24,614
netdiversion	129,180	119,589	615,282
ns1688347	2,685	4,191	66,908
opm2-z7-s2	2,023	31,798	79,762
reblock67	670	2,523	7,495
rmine6	1,096	7,078	18,084
sp98ic	10,894	825	316,317
tanglegram1	34,759	68,342	205,026
tanglegram2	4,714	8,980	26,940
vpphard	51,471	47,280	372,305

5 Experiments and Results

The proposed heuristic in this work reads a series of test instances (initially applied to binary problems, as described in the previous section), the fractional solution and an initial integer feasible solution to the problem. This initial solution was generated using Feasibility Pump and is informed as initial solution for all tested MIP heuristics.

For most of the practical operations research applications, solution methods are only useful if they are able to produce satisfatory solutions in short periods of time. Thus, we imposed a time limit of 300 seconds. The initial size parameter of the problem to be solved in these tests was defined as 100 (the number of free variables). Another parameter is the percentage of increase in the size of the problem(in this case, it was used 50). The tests were run using the following hardware configuration: Intel (R) Core (TM) i7 CPU, 1.90GHz, 6 GB RAM.

Preliminary results described below, consider the implementation of the method RINS adapted as proposed in this work. The instances used in the tests are described in Table 1.

We compare our results with the standalone CBC solver as well, with an implementation method RINS in original form. Both CBC and RINS use the same initial solution that our method uses.

In Table 2 the values found for each instances are described, considering the implementations used. For each instance, the best value found is highlighted in bold. Comparing the values obtained, we can notice that the proposed method achieves better results on 14 instances when compared to CBC and is better in 18 instances when compared to the original form of RINS. In 15 instances, the method obtains the same value achieved when the problem is solved using the CBC, while the result is the same in 15 cases when compared to the original RINS.

Tables 3 and 4 present the results considering the number of wins, draws and defeats comparing the implementations tested. In all groups of instances, the proposed method has achieved good results, reaching better solutions than or equal to the comparison method. For example, the group of MIPLib instance, the pRINS obtains 76% of equal or better values when compared to CBC, and 88% compared to RINS.

Table 5 shows the results for each tested implementation considering the sum and the average of gaps (percentage difference between the obtained value and the best known value). The proposed method was the one that was closest, on average, to the best values. The metric gap solution was calculated according to the following expression: $\min(100, (z - best) \div best \times 100)$.

Table 2. Results

Instances	pRINS		CBC		RINS		Best
	z	gap %	z	gap %	z	gap %	z
long01	**161**	0.0	**161**	0.0	**161**	0.0	161
long-hidden01	2,745	100.0	2,747	100.0	**2,606**	100.0	346
long-hint01	**74**	0.0	**74**	0.0	**74**	0.0	74
long-late01	**2,895**	100.0	**2,895**	100.0	**2,895**	100.0	235
medium01	**374**	0.0	383	2.4	383	2.4	374
medium-hidden01	**1,287**	100.0	**1,287**	100.0	**1,287**	100.0	111
medium-hint01	**245**	0.0	255	4.0	250	2.0	245
medium-late01	**779**	100.0	970	100.0	966	100.0	157
sprint01	91	22.9	**74**	0.0	96	29.7	74
sprint-hidden01	92	100.0	**80**	100.0	99	100.0	32
sprint-hint01	**297**	0.0	400	34.6	496	67.0	297
sprint-late01	**74**	0.0	81	9.4	84	13.5	74
air04	**56,137**	0.0	56,138	0.0	60,093	7.0	56,137
bley-xl1	230	21.0	230	21.0	**215**	13.1	190
cov1075	**20**	0.0	**20**	0.0	**20**	0.0	20
eil33-2	1,011	8.2	**1,000.24**	7.0	1,858	98.9	934
eilB101	2,093	72.1	**1,529.88**	25.7	2,298	88.9	1216
iis-100-0-cov	30	3.4	**29**	0.0	32	10.3	29
iis-bupa-cov	**37**	2.7	38	5.5	38	5.5	36
iss-pima-cov	34	3.0	34	3.0	**33**	0.0	33
macrophage	**447**	19.5	809	100.0	596	59.3	374
mine-166-5	**-3.98740e+08**	29.6	**-3.98740e+08**	29.6	**-3.98740e+08**	29.6	-5.66396e+08
mine-90-10	**-3.98740e+08**	49.1	**-3.98740e+08**	49.1	**-3.98740e+08**	49.1	-7.84302e+08
n3div36	**136,000**	3.9	148,600	13.6	149,800	14.5	130,800
n3seq24	73,200	40.2	**62,800**	20.3	69,400	32.9	52,200
neos-1109824	467	23.5	**380**	0.5	760	100.0	378
neos-1337307	-201,447	0.4	**-201,466**	0.4	-201,447	0.4	-202319
neos18	**16**	0.0	**16**	0.0	**16**	0.0	16
netdiversion	**370**	52.8	451	86.3	451	86.3	242
ns1688347	**35**	29.6	**35**	29.6	**35**	29.6	27
opm2-z7-s2	-1,317	87.1	**-9,365**	8.9	-1,317	87.1	-10,280
reblock67	**-2.19704+e07**	36.5	**-2.19704+e07**	36.5	**-2.19704+e07**	36.5	-3.46306+e07
rmine6	**-449.25**	1.7	**-449.25**	1.7	**-449.25**	1.7	-457.18
sp98ic	**4.50307+e08**	0.2	4.87071+e08	8.4	4.68947+e08	4.4	4.49145+e08
tanglegram1	**5,494**	6.0	**5,494**	6.0	**5,494**	6.0	5182
tanglegram2	**443**	0.0	**443**	0.0	**443**	0.0	443
vpphard	**16**	68.7	22	100.0	22	100.0	5

Table 3. Number of victories and defeats for each group of instances

Group	Best Value		
	pRINS	Equal	CBC
MIPLIB library	8	11	6
nurse scheduling	6	4	2

Table 4. Number of victories and defeats for each group of instances

Group	Best Value		
	pRINS	Equal	RINS
MIPLIB library	11	11	3
nurse scheduling	7	4	1

Table 5. Sum and average of gaps

	p-RINS	CBC	RINS
Sum	1,082.10	1,103.52	1,475.70
Average	29.24	29.82	39.88

6 Final Remarks

Even tough this work is still being developed, encouraging results were obtained for the proposed RINS variant, denoted here as pRINS. Adjustments are being made in the algorithm to speed up the execution of multiple pre-processing phases, which can produce further speedups. Our computational results show that pRINS is already better or equal to other methods (using only CBC or original RINS) in most cases, considering the production good feasible solutions in a restricted period of time.

References

[1] F. Glover Tabu Search and adaptive memory programming - advances, applications and challenges Interfaces in Computer Sciences and Operations Research 1-75 (1996)

[2] C.R. Reeves Genetic Algorithms Modern Heuristic Techniques for Combinatorial Problems, Advanced Topics in Computer Science Series chapter 4, pages 151-196. Blackwell Scientific Publications, (1993)
[3] D.E. Goldberg Genetic Algorithms in Search, Optimization and Machine Learning Addison-Wesley (1989)
[4] F. Glover Future paths for Integer Programming and links to Artificial Intelligence COR Vol. 13, No. 5, 533-549 (1986)
[5] C. Blum and J. Puchingerb and G. Raidl and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey Applied Soft Computing 11:4135-4151 (2011)
[6] J. Eckstein and M. Nediak Pivot, Cut, and Dive: a heuristic for 0-1 mixed integer programming Journal Heuristics 13:471-503 (2007)
[7] F. Parisini and M. Milano Improving CP-based Local Branching via Sliced Neighborhood Search Proceedings of the 2011 ACM Symposium on Applied Computing 887-892 (2011)
[8] S Ghosh DINS, a MIP Improvement Heuristic International IPCO Conference 310-323(2007)
[9] T Berthold Rens: The Relaxation Enforced Neighborhood Search (2009)
[10] D Ferreira and R Morabito and S Rangel Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants Computers e Operations Research 37:684-691 (2009)
[11] E. Danna and E. Rothberg and C. Pape Exploring relaxation induced neighborhoods to improve MIP solutions 102:71-90 (2005)
[12] M. Fischetti, L. Bertacco, and A. Lodi. A feasibility pump heuristic for general mixed- integer problems. Technical report, Universit'a di Bologna D.E.I.S. Operations Research (2005).
[13] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. Mathematical Programming, 104(1):9104, (2005).
[14] Fischetti, M. e A. Lodi Local branching, Mathematics Programming, ser. B 98, p. 23-47 (2003).
[15] Rothberg E. An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. INFORMS Journal on Computing Vol.19, No.4, pp.534-541(2007).
[16] Forrest, J. and Lougee-Heimer, R. INFORMS Tutorials in Operations Research. p.257-277. CBC User Guide (2005).
[17] Lougee-Heimer, R. IBM Journal of Research and Development. The Common Optimization Interface for Operations Research: Promoting open-source software in the operations research community. Vol.47 No.1 p.57-66 (2003)
[18] N. Mladenovic and P. Hansen. Variable Neighborhood Search. Computers and Operations Research, 24:1097-1100 (1997).
[19] Haspeslagh, S. and De Causmaecker, P. and Stolevik, M. and A., Schaerf. First international nurse rostering competition 2010. CODeS, Department of Computer Science. KULeuven Campus Kortrijk. Belgium (2010)
[20] Koch, T. and Achterberg, T. and Andersen, E. and Bastert, O. and Berthold, T. and Bixby, R. and Danna, E. and Gamrath, G. and Gleixner, A. and Heinz, S. and Lodi, A. and Mittelmann, H. and Ralphs, T. and Salvagnin, D. and Steffy, D. and Wolter, K. Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany. MIPLIB 2010. Mathematical Programming Computation. Mathematics and Statistics. Springer Berlin/Heidelberg Vol.3 p.103-163 In http://dx.doi.org/10.1007/s12532-011-0025-9 (2011)