Genetic Programming for Guiding Branch and Bound Search

Konstantinos Kostikas and Charalambos Fragakis

Department of Computational Mathematics and Computer Programming
School of Mathematics, Physics and Computational Sciences
Faculty of Technology
Aristotle University of Thessaloniki
54006 Thessaloniki, Greece
{kostikas, fragakis}@gen.auth.gr

Abstract. We propose how Genetic Programming (GP) can be used for developing, in real time, problem-specific heuristics for Branch and Bound (B&B) search. A GP run, embedded into the B&B process, exploits the characteristics of the particular problem being solved, evolving a problem-specific heuristic expression. The evolved heuristic replaces the default one for the rest of the B&B search. The application of our method to node selection for B&B based Mixed Integer Programming is illustrated by incorporating the GP node selection heuristic generator into a B&B MIP solver. The hybrid system compares well with the unmodified solver utilizing DFS, BFS, or even the advanced Best Projection heuristic when confronted with hard MIP problems from the MIPLIB3 benchmarking suite.

1 Introduction

The solution of large scale hard combinatorial optimization problems is one of the areas where today's state of the art algorithms and computer machinery reach their limits. Logistics, high tech manufacturing and computational biology are only but a few of the application domains where such problems are routinely encountered. Consequently, improved algorithmic techniques which enable the execution of even larger problem instances are highly desirable.

Optimal or good enough solutions to problems of combinatorial nature can be obtained by applying exact algorithmic methods as well as non-exact techniques like Genetic Algorithms and Tabu Search, frequently described as metaheuristics. Often the combination of these two seemingly distinct approaches, that is exact with non-exact methods, yields the best results [1]. For example, in Branch-and-Bound (B&B), the most widely used technique for solving combinatorial optimization problems, metaheuristics are repeatedly used for obtaining fast a feasible solution, and thus for providing the algorithm with a lower bound, which is exploited for speeding up the search.

Apart from how fast a good early lower bound is found, two other sources of uncertainty exist in all branch and bound algorithms: branching, which decides how the solution space is subdivided into subproblems, and node selection. Both of them greatly affect how efficiently the solution space is traversed. For that reason, heuristic functions are often used, trying to exploit problem domain specific knowledge in order to perform educated guesses as to where the search should be directed to next.

Branching and node selection heuristics, although extremely efficient at times, are not however guaranteed to perform well in all cases. In their study on heuristics for B&B Mixed Integer Programming, Linderoth and Savelsbergh demonstrate for example that no branching or node selection method outperforms all others when a variety of MIP problems is concerned [2]. The reason for that is that heuristic functions, however robust, cannot be equally effective in all problem instances. This gave rise to our idea: what if we employ the power of metaheuristics and machine learning for assisting B&B decision making by evolving customized, problem instance-specific heuristics?

In this paper we present how Genetic Programming (GP) [3], an evolutionary computation technique with rich expressive power, can be applied to the automatic generation of node selection heuristics for B&B MIP. Static, as well as dynamic, that is in runtime , evolution of node selection expressions via GP is discussed and a prototype realization of the latter on B&B based MIP is demonstrated.

The rest of this paper proceeds as follows: The remaining of this section briefly outlines related work. Section 2 is an introduction to Branch and Bound in general and to B&B based MIP in particular. Section 3 talks about how GP can be integrated into B&B for the automated evolution of heuristics and the issues related with the construction of the training set. Section 4 briefly discusses generation of B&B heuristics in vitro, Sect. 5 describes the experimental parameters of our study, Sect. 6 presents the obtained results, and Sect. 7 summarizes our conclusions.

1.1 Related Work

The field of metaheuristics for the application to combinatorial optimization problems is a rapidly growing field of research. A recent, general “overview and conceptual comparison” is [4]. Abramson and Randall [5][6] utilize Simulated Annealing, Tabu Search and other metaheuristics for building a ’general purpose combinatorial optimization problem solver’; [6] and [1] are also good references for related work. Mitchell and Lee [7] cite several references regarding the application of metaheuristics in MIP.

As far as the automatic generation of B&B heuristics is concerned, it was mentioned to us that the possibility of applying learning techniques in branch selection for Integer Programming was first suggested by Glover in [8][1], with no implementation. However, to our knowledge, there exists no previous attempt of utilizing metaheuristics or evolutionary computation techniques for evolving

[1] We didn’t manage to get hold of the actual paper.

node selection methods. We are also not aware of previous uses of Genetic Programming in MIP.

2 Background

2.1 Branch and Bound

Branch and Bound is more of a framework than a specific algorithm; this was firstly recognized by Lawler and Wood in [9], which contains also a problem independent description of B&B. A presentation of the most important models of B&B and the related references can be found in [10]. Although the B&B framework is not tied to any particular description, B&B methods are most often described as generating search trees [11]: roughly speaking, each node corresponds to a subset of the feasible solution set. A subproblem associated with a node is either solved directly, or its solution set is split, and for each subset a new node is added to the tree. The process is improved by computing a bound on the solution value a node can produce. If the bound is worse than the value of the best solution found so far, the node cannot produce a better solution, and, hence, it can be excluded from further examination.

B&B is an enumeration method; under most circumstances it is guaranteed that the optimal solution will be eventually found. However, as it was already mentioned, the choices made when the algorithm is branching or selecting a node for visiting next make a big difference on how fast this will happen. For that reason, heuristics are used for predicting the outcome of these choices. Especially in cases where the amount of computation performed in each node of the tree is significant, as is the case in B&B MIP, it pays off to spent some time estimating what node should be visited next.

2.2 B&B based Mixed Integer Programming

A mixed integer program (MIP) is an optimization problem stated mathematically as follows:

$$Maximize \qquad z_{MIP} = \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \tag{1}$$

$$subject\ to \qquad \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \quad i = 1, \ldots m \tag{2}$$

$$\begin{aligned} l_j \leq x_j \leq u_j &\qquad j \in N \\ x_j \in Z &\qquad j \in I \\ x_j \in R &\qquad j \in C, \end{aligned}$$

where I is the set of integer variables, C is the set of continuous variables, and $N = I \cup C$. The lower and upper bounds l_j and u_j may take on the values of plus or minus infinity. Thus, a MIP is a linear program (LP) plus some integrality restriction on some or all of the variables [2].

Although other methods also exist, B&B is the predominant technique for solving MIP, employed in most commercial solvers [2]. To precisely define how B&B can be applied for solving MIP problems, some definitions are needed. We use the term node or subproblem to denote the problem associated with a certain portion of the feasible region of MIP. Define z_L to be a lower bound on the value of z_{MIP}. For a node N^i, let z_U^i be an upper bound on the value that z_{MIP} can have in N^i. The list $\mathcal{L}$ of problems that must still be solved is called the active set. Denote the optimal solution by x^*. The algorithm in Table 1, adopted from [2], is a Linear Programming-based Branch and Bound algorithm for solving MIP.

Table 1. Branch and Bound Algorithm for Linear Programming based MIP

```
0. Initialize. $\mathcal{L}$ = MIP. $z_L = -\infty$. $x^* = \emptyset$.
1. Terminate? Is $\mathcal{L} = \emptyset$? If so, the solution $x^*$ is optimal.
2. Select. Choose and delete a problem $N^i$ from $\mathcal{L}$.
3. Evaluate. Solve the LP relaxation of $N^i$. If the problem is infeasible, go to
   step 1, else let $z_{LP}^i$ be its objective function value and $x^i$ be its solution.
4. Prune. If $z_{LP}^i \leq z_L$, go to step 1. If $x^i$ is fractional, go to step 5, else let
   $z_L = z_{LP}^i$, $x^* = x^i$, and delete from $\mathcal{L}$ all problems with $z_U^j \leq z_L$.
   Go to step 1.
5. Divide. Divide the feasible region of $N^i$ into a number of smaller feasible
   regions $N^{i1}$, $N^{i2}$, ..., $N^{ik}$ such that $\cup_{j-1}^k N^{ij} = N^i$. For each $j = 1, 2, \ldots, k$,
   let $z_U^{ij} = z_{LP}^i$ and add the problem $N^{ij}$ to $\mathcal{L}$. Go to 1.
```

2.3 Node Selection in B&B MIP

A plethora of methods, or "strategies", exist for node selection in MIP. [2] is a good survey of lots of them. All such methods, except from pure DFS and BFS, employ in their core an expression which assigns a numerical value to all active nodes of the tree. The node which achieves the highest score is selected by B&B for being visited next. This rating is often the estimated value of the best obtainable solution from the specific node. Characteristic example of an estimation method is the popular Best Projection method [12], which is employed in GLPK, the MIP solver we used for our experimentation (see Sect. 5.1) as the default node selection heuristic. In the next section we explain how GP can be utilized for evolving problem domain or problem instance specific node selection expressions.

3 Evolving B&B Heuristics using Genetic Programming

If one was able to devise a heuristic method customized for the specific problem at hand, maybe considerable performance gains could be realized. Manually building dataset-specific heuristics doesn't make practical and economical sense;

however, GP offers the means for doing exactly that: By embedding in the B&B algorithm a Genetic Programming run, customized heuristics can be evolved, which will be consequently used for the rest of the B&B search. In other words, our suggestion is that GP can be used for evolving problem instance specific heuristics in real time. Obviously, Genetic Programming is not the only machine learning technique capable for performing this task; it is however highly convenient, because it directly evolves executable expressions which can be used as heuristics without further modification. The approach consists of three distinct stages:

B&B Stage1 begins with the start of the search and lasts until a criterion is met, i.e. until a specified number of solutions have been found, or until a specified number of nodes have been visited[2]. Stage 1 is ordinary B&B search, employing standard heuristics if necessary, with the addition that some extra data needs to be maintained, reflecting the progress of the search.

GP Stage is where the GP search takes place: Initially the training set is constructed (Sect. 3.2), and after that the GP run is performed. At the end of the run, the best evolved expression, that is the fittest individual, is selected and becomes the heuristic to be used for the rest of the B&B search. If a static heuristic was used in B&B Stage1, this is replaced by the evolved one.

B&B Stage2 resumes the execution of B&B, which was paused in the previous stage, and continues the search by utilizing the GP evolved expression for guiding the B&B search.

Further GP Stages. In our prototype implementation, the loop starting with B&B Stage1 and finishing with B&B Stage2 is executed once, i.e. only one GP Stage is used during the life-cycle of the B&B search. We believe further GP stages would improve the quality of the evolved heuristics significantly: in many problem domains the 'structure' of the search space changes dramatically as the search proceeds [1]. In such cases periodic GP stages might be necessary in order to 'maintain' efficient heuristics. This is also related with the quality of the GP training set, see Sect. 3.2.

Finally, it is worth pointing out that the above approach can be used for evolving more than one type of heuristics at the same time, for example node selection as well as branching heuristics.

3.1 Building Blocks

Since heuristics make use of the special characteristics and the distinct attributes of the problem domain, the choice of the terminal set needs obviously to reflect

[2] Or, in our case, until the specified number of "dives" have been performed, see Sect. 3.2.

the nature of the problem. In our experimental implementation of a GP-enhanced B&B MIP solver, the terminals used are common primitives of node selection methods for MIP, like for example *lp_solN*$_i$ which is the upper bound at the 'parent' of node N^i. The terminals we used are listed in Table 2. They are quite simple in comparison to the ingredients of advanced MIP node selection methods. More advanced terminals would probably improve the capabilities of our GP-enhanced solver.

Table 2. GP Terminals for node selection method construction. *Constant Terminals* involve general characteristics of the MIP problem and information regarding the solution of the LP relaxation of the problem. *Dynamic Terminals* concern the specific B&B node being evaluated.

Property	**Description**
	Dynamic Terminals
ii_sumN$_i$	$s^i \equiv \sum_{j \in I} min\left(x_j^i - \lfloor x_j^i \rfloor, 1 - \left(x_j^i - \lfloor x_j^i \rfloor\right)\right)$ Sum of integer infeasibilities of the relaxation solution z_U^i at the parent of node N^i
ii_countN$_i$	Count of integer infeasibilities at the parent of node N$_i$
lp_solN$_i$	Value of the solution of the linear programming relaxation of the parent of node N$_i$
tree_depth_N$_i$	Depth of the tree at node N$_i$
	Constant Terminals
intvar_count	Number of integer variables of the problem (including binary variables)
binvar_count	Number of binary variables of the problem
totvar_count	Total number of variables of the problem (integer and continuous)
ii_sumN$_0$	Sum of integer infeasibilities at the root node of the Branch and Bound tree (i.e. of the linear relaxation solution of the MIP problem)
ii_countN$_0$	Count of integer infeasibilities at the root node of the Branch and Bound tree (i.e. of the linear relaxation solution of the MIP problem)
lp_solN$_0$	Value of the solution of the linear programming relaxation of the MIP problem
bonly	TRUE if the problem contains only Binary integer variables, FALSE otherwise

3.2 Training Sets

Evolving a problem-specific node selection method using Genetic Programming means that a problem-specific training set is required as well. The training set has to make use of data generated during B&B Stage1, in order to allow GP to evolve a node selection method which will successfully 'guide' the B&B search in Stage2. The scheme we devised for automatically constructing the training cases is based on the observation that B&B search can be thought of as a collection of "dives": each dive d^i is a path starting at a non-leaf node of the B&B tree and finishing at a leaf node. Dives are formed in a sequential manner, in case of DFS based exploration, or in parallel in any other case. Each dive performed is essentially an attempt of the search to reach a solution. Given the above, our training set is constructed in GP Stage as follows:

1. The dive is re-created, based on node-state information collected in B&B Stage1.
2. It is decided if the dive will enter the training set or not.

3. If yes, it is assigned a score, by applying to it a rating function R and it is inserted into the training set.

Of course, the problem is how to rate each dive: in our experimentation we used the following rating function:

$$R_i = \left(\frac{s^i_{start} - s^i_{finish}}{d^i_{finish} - d^i_{start}} \right) V_{bonus} \tag{3}$$

where s^i_{start} is the sum of integer infeasibilities (see Table 2) at the node where the dive starts, s^i_{finish} at the node where it ends, d^i_{finish} and d^i_{start} is the depth of the B&B tree at the finishing and starting node respectively, and V_{bonus} is a bonus factor applied if the dive results in a full integer feasible MIP solution. Our rational was that fast reduction of integer infeasibility denotes a promising dive; in addition, dives resulting in full integer feasible solutions deserve to be rewarded with the V_{bonus} factor[3]. Finally, all dives performed in B&B Stage1 were inserted into the training set, except from the length-one dives whose unique (leaf) node was an unfeasible LP relaxation.

We believe that the above scheme, used by us in a rather straightforward manner, is quite powerful: For one, sophisticated domain specific rating methods can be utilized. Even more important, arbitrary metrics, like for example cumulative CPU time spent at each dive or main memory allocated, can be used. This appears especially appealing to us because it allows the generation of heuristics tuned for the particular combination of hardware/software platform, algorithmic implementation and problem data set at hand. Such customized heuristics are probably beyond the reach of a human-originated design.

4 Evolving Heuristics Offline

Genetic Programming has been used in standalone mode for heuristic generation in domains other than B&B search. A representative recent example is [13] where GP is used "to optimize the priority functions associated with register allocation as well as branch removal via predication". A number of benchmark programs were used as the training set. Such a route for heuristic generation is probably applicable to B&B as well. In the case of B&B MIP, benchmark problems from MIPLIB3 [14] like the ones used for our experimentation (Sect. 6) can be utilized as the training set.

An issue with the above approach is the excessive computational time required. However, in the case of B&B heuristics at least, 'brute force' does not need to be the only choice: as discussed in Sect. 2.3, common node selection heuristics in B&B MIP are expressions that provide an estimate of the best solution obtainable from each node of the tree. In an experiment, we calculated the best solution obtainable from each one of the 984 nodes comprising the B&B tree formed during the solution of problem MISC03 of MIPLIB3 by our MIP

[3] For all the experiments we used $V_{bonus} = 3.0$ for integer feasible solutions worst than the lower bound and $V_{bonus} = 5.0$ for solutions better than the lower bound.

solver, GLPK; the calculated values were used as the training set[4] in GP runs aimed at evolving node selection heuristics. The same GP control parameters as for dynamic heuristic generation were used (Tables 2 and 3), except from a reduced function set. Although not extended experimentation was performed, the evolved heuristics seemed to constantly outperform the Best Projection node selection method at predicting the best solution obtainable from a node. Similar trials performed on MIP problem formulations for Minimax Robust Regression Estimators [15] produced also positive results. Further experimentation is of course required in order to assess the method.

5 Experimentation Setup

5.1 Infrastructure

The MIP solver we based our experimentation on is GLPK (Gnu Linear Programming Kit) [16]. GLPK doesn't provide the plethora of options found in commercial software like CPLEX [17][18], but it contains a solid implementation of the simplex method, and, most importantly, comes with full source code. GLPK adopts a backtracking method for node selection: it goes depth first as much as possible, and then backtracks by selecting a node using DFS[5], Breadth-First-Search (BFS), or Best Projection. The necessary hooks were placed into GLPK in order to cater for collecting data during B&B Stage1, for performing the GP run, and for replacing the default backtracking method with the one evolved. Except from the node selection method utilized, the default settings were applied in all cases. For Genetic Programming we used strongly typed lilgp[19][20], which was integrated with the GLPK infrastructure.

5.2 GP Run Parameters

The GP parameters we used in our tests are listed in Table 3. GP Stage evolves 1000 individuals for 50 generations. All individuals constituting the first generation are randomly created, and each subsequent generation is formed using individuals resulting from crossover operations, with 88% probability, mutation, with 10% probability, and reproduction with 2% probability. Tournament selection with size seven is used. Each individual is restricted to 75 nodes, with no restriction to tree depth. Finally, the standard arithmetic and boolean logical and comparison primitives listed in Table 3 are used.

6 Results

In our experimentation, our goal was twofold. For one, we wanted to find out if the hybrid system, containing GP as node selection heuristic generator, would

[4] Actually 685 training cases out of the 984 nodes were made because at the remaining nodes no integer feasible solution could be found.

[5] DFS selects a node using LIFO: this results in B&B spending most of its time evaluating nodes found to the bottom of the tree; such a strategy is good at finding solutions, but is likely to get stuck in specific areas of the search space.

Table 3. Control Parameters for B&B Stage1 and the GP Stage.

Objective:	evolve LP based Branch and Bound node selection heuristic specialized for MIP problem instance
Function set:	ADD SUB MUL PDIV AND OR NOT IFTRUE IFGTE IFLTE IFEQ
Terminal set:	B&B and LP related runtime data, see Table 2
Fitness cases:	dynamically created based on data obtained in B&B Stage1 (5, 15 and 35 dives)
Fitness function:	standardized fitness, based on mean error over the fitness cases
Population size:	1000
Initial population:	initialization method: full (50%), grow (50%) initial depth: 4-6
Crossover probability:	88 percent
Mutation probability:	10 percent
Selection:	fitness-proportionate
Termination:	generation 50
Maximum nodes of tree:	75
Parameters for B&B Stage1:	stage ends when 5, 15 or 35 dives have been performed

be able to compete with standard GLPK using BFS and DFS node selection methods, or the advanced Best Projection heuristic. In addition, we wanted to assess how the size of the training set would affect the quality of the generated heuristic, and thus its the capability to effectively guide B&B towards good solutions.

For the above purposes we used for experimentation problems from the standard MIPLIB3 library [14]. Easy (fully solved to optimality in less than 300 seconds by the unmodified solver) and very difficult problems (no solution of the initial LP relaxation found in the available time) were excluded. In addition, problems with small B&B search trees were excluded, because not enough dives for building the training set were performed in B&B Stage1. The remaining 13 problems were presented to unmodified GLPK, utilizing best projection, BFS and DFS as the node selection method, and to the GP-enhanced GLPK, using training sets of 5, 15 and 35 dives. The time available for each run was set to 300 seconds. 5 runs were performed under each case for standard GLPK, and 25 runs (5 repetitions for 5 random initial populations) for the hybrid solver. All tests were performed in the same hardware/software system (Pentium 4 at 2.4GHz with 768Mb Ram running Linux 2.4). The best (minimum, since all are minimization problems) solution obtained by each method is presented in Table 4. For the GP based methods, the mode of the solutions obtained is reported.

As it is apparent from Table 4, no node selection method fully outperforms the others. This is typical for node selection heuristics, and in accordance with [2]. Two methods stand out however as being the most capable of obtaining good integer feasible solutions at the available time: best projection and GP-35 (training set comprised of 35 dives): best projection was able to find the best overall solution in 6 cases, whereas GP-35 in 5 cases. In the remaining two cases, problems modglob and p2756, the best solutions were found by GP-5 and DFS respectively.

Where the GP evolved heuristics stand our however, is in their consistency: All GP evolved heuristics managed to acquire solutions in all problems in all the

Table 4. Best solutions found in the available time (300 seconds). Bestp, BFS and DFS are standard node selection methods. GP35, GP15 and GP5 are methods dynamically evolved by GP using a training set of 35, 15 and 5 cases (dives) respectively. 'NF' means that no solution was found. Overall best values are printed bold with the faster method preferred in case of ties. For the GP methods, the mode of the solutions obtained is reported, or the median value (in italics) if no mode exists.

Problem	Bestp	BFS	DFS	GP35	GP15	GP5
10teams	928	948	934	**926**	928	984
fiber	**415629**	426301	531856	524332	*447136*	426301
gesa2	**25977000**	26401200	26523800	*26541200*	26423100	26402600
gesa2$_o$	26070200	26243900	26416200	**26035700**	*26282500*	26314900
gt2	31582	21166	31023	**21166**	*31023*	31023
mod011	**-53548200**	-49444800	-48489400	-52369800	-50108800	-45495900
modglob	20931300	21033600	21442800	20992000	21054400	**20920400**
p2756	NF	NF	**3844**	4222	*26267*	*10532.5*
pk1	**11**	17	17	16	17	17
qiu	-132.873	-27.6516	102.332	**-132.873**	-119.654	-14.4329
rout	NF	NF	1353.75	**1167.17**	*1213.33*	*1231.25*
set1ch	**60342.8**	63350	64744.2	63373.5	64744.2	64744.2
vpm2	**13.75**	14.5	14.25	14.5	15	14.5
Best Sols	**6**	**-**	**1**	**5**	**-**	**1**
Avg Rank	**2.54**	**3.62**	**4.69**	**2.46**	**3.62**	**4.23**

runs that were performed, whereas Best Projection and BFS found no solution at all in two of them, p2756 and rout. DFS found also solutions to all problems, although not as good as the solutions found by the GP methods, especially by GP35.

The obtained results are also quite illustrative on how the size of the training set, that is of the number of dives performed in B&B Stage1, affects the the quality of the heuristic evolved; GP-35, enjoying a bigger training set in GP-Stage consisting of 35 training cases, evolves significantly better heuristics than GP-15 and GP-5 which employ 15 and 5 training cases respectively. This is reflected in the average rank of the methods, shown in Table 4.

Table 5. Best and worst solutions obtained for the GP-enhanced MIP solver.

	GP-35		GP-15		GP-5	
Problem	Best Run	Worst Run	Best Run	Worst Run	Best Run	Worst Run
10teams	924	928	928	928	934	984
fiber	524299	558087	418318	1153830	418318	1104800
gesa2	26502300	26557500	26363000	27338200	26402600	26746300
gesa2$_o$	26035700	26037800	26258000	26373100	26078000	26548500
gt2	21166	35328	21166	37142	31023	31023
mod011	-52632100	-52369800	-50108800	-50108800	-45659000	-45495900
modglob	20920400	21240400	20895000	21054400	20920400	21400300
p2756	4222	48124	15520	36249	3505	73564
pk1	16	17	14	17	17	17
qiu	-132.873	-115.248	-119.654	-119.654	-132.873	33.0187
rout	1077.56	1270.63	1142.75	1318.13	1167.17	1367.94
set1ch	63373.5	63373.5	62426	64744.2	64744.2	64744.2
vpm2	14.5	14.5	14.5	15	14	15.5

Table 6. Average time in seconds used for B&B Stage1 and for GP Stage. The time spend in each stage increases in proportion to the size of the training set.

	GP-35		GP-15		GP-5	
Problem	B&B Stage1	GP Stage	B&B Stage1	GP Stage	B&B Stage1	GP Stage
10teams	117.30	5.73	69.58	2.91	24.12	1.59
fiber	9.96	5.78	5.56	2.98	2.54	1.62
gesa2	12.14	4.98	4.75	2.83	2.53	1.62
gesa2$_o$	10.30	5.82	3.32	3.24	1.66	1.62
gt2	0.27	5.58	0.20	2.98	0.18	1.70
mod011	146.04	6.19	81.95	3.03	42.75	1.70
modglob	2.38	6.08	1.39	3.25	0.94	1.64
p2756	5.11	5.69	2.53	2.65	0.53	1.46
pk1	0.70	5.71	0.29	2.95	0.12	1.52
qiu	16.72	5.91	9.78	3.04	5.02	1.63
rout	13.38	6.09	4.75	3.16	1.51	1.65
set1ch	0.97	5.40	0.86	2.69	0.80	1.18
vpm2	1.98	5.17	0.97	2.39	0.38	1.31

Table 5 depicts the best and worst solutions obtained by each GP setup and Table 6 shows how the size of the training set affects the time spent in B&B Stage1 and in GP Stage. It is worth mentioning that GP-35, supported by a larger training set, manages to do best despite the significantly less time spent in B&B Stage2[6], in comparison to all other methods.

7 Conclusions and Further Research

We used Genetic Programming as a component in a Branch and Bound framework, where GP is utilized for generating the node selection heuristic for MIP. We believe that the experimental results obtained by our prototype implementation show that the hybrid B&B-GP approach we introduce portrays significant potential: supported by a properly constructed training set of adequate size, problem-instance specific heuristics can be evolved, capable of consistently guiding B&B towards promising areas of the search space.

Concerning our future research efforts, these will be directed in two fronts: The first one is to incorporate multiple GP Stages in our design, as well as to experiment with more elaborate GP structures and techniques like ADFs and Interval Arithmetic. The second one will be to increase our understanding of how the training set construction method adopted in the GP Stage affects the search in B&B Stage2. In addition to the above, we would like to apply our approach to more domains where B&B heuristic-based search is used.

References

1. Michalewicz, Z., Fogel, D.B.: How to Solve it: Modern Heuristics. Springer-Verlag (2002)

[6] The duration of B&B Stage2 equals the total run time minus the duration of B&B Stage1 and of GP Stage.

2. Linderoth, J., Savelsbergh, M.: A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing **11** (1999) 173 – 187
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys **35** (2003) 268–308
5. Abramson, D., Randall, M.: A simulated annealing code for general integer linear programs. Annals of Operations Research **86** (1999) 3–24
6. Randall, M., Abramson, D.: A general metaheuristic based solver for combinatorial optimisation problems. Kluwer Journal on Computational Optimization and Applications **20** (2001)
7. Mitchell, J., Lee, E.K.: Branch-and-bound methods for integer programming. In Floudas, C.A., Pardalos, P.M., eds.: Encyclopedia of Optimization, Kluwer Academic Publishers (2001)
8. Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers and Operations Research (1986) 533–549
9. Lawler, E., Wood, D.: Branch-and-bound methods: a survey. Operations Research **14** (1966) 699–719
10. A. de Bruin, G.A.P. Kindervater, H.T.: Towards an abstract parallel branch and bound machine. Technical report, Erasmus University, Department of Computer Science (1995)
11. Robert, S.: Algorithms. Addison-Wesley (1983)
12. Mitra, G.: Investigation of some branch and bound strategies for the solution of mixed integer linear programs. Mathematical Programming **4** (1973) 155–173
13. Stephenson, M., O'Reilly, U.M., Martin, M.C., Amarasinghe, S.: Genetic programming applied to compiler heuristic optimization. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E., eds.: Genetic Programming, Proceedings of EuroGP'2003. Volume 2610 of LNCS., Essex, Springer-Verlag (2003) 245–257
14. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.P.: An updated mixed integer programming library: MIPLIB 3.0. Optima **58** (1998) 12–15
15. Zioutas, G.: Quadratic mixed integer programming models in minimax robust regression estimators. In: Statistics for Industry and Technology. Verlag (2004)
16. GLPK: (www.gnu.org/software/glpk)
17. CPLEX: (www.ilog.com/products/cplex)
18. Bixby, R., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and practice – closing the gap. In: System Modelling and Optimization: Methods, Theory, and Applications. Volume 174 of IFIP INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING. Kluwer Academic Publishers, Boston (2000) 19–49
19. Zongker, D., Punch, B.: lilgp 1.01 user's manual. Technical report, Michigan State University, USA (1996)
20. Luke, S.: (Strongly typed lilgp, www.cs.umd.edu/users/seanl/gp/patched-gp/)