2015 IEEE Symposium Series on Computational Intelligence

An Evolutionary Approach to the Discovery of Hybrid Branching Rules for Mixed Integer Solvers

Kjartan Brjánn Pétursson
School of Engineering and Natural Science
University of Iceland
Reykjavik, Iceland
Email: kbp4@hi.is

Thomas Philip Runarsson
School of Engineering and Natural Science
University of Iceland
Reykjavik, Iceland
Email: tpr@hi.is

***Abstract*—An evolutionary algorithm is used to search for problem specific branching rules within the branch-and-bound framework. For this purpose an instance generator is used to create training data for an integer programming problem, in particular the multi-dimensional 0/1 knapsack problem. An extensive experimental study will illustrate that new and more effective rules can be found using evolutionary computation.**

I. INTRODUCTION

The key motivation for this work is to use evolutionary algorithms to design and improve search algorithms, rather than designing evolutionary algorithms to solve the problems directly.

An effective approach to solving integer programming problems is using a mixed integer programming (MIP) solver. MIPs, in general, are commonly solved by a dedicated MIP-solver, employing a *branch-and-bound* algorithm. One approach to improving the efficiency of the branch-and-bound algorithm is by using an effective branch selection rule. In this work we will try to improve this rule for a specific integer programming problem. The approach taken will be to apply an evolutionary algorithm to search for a better rule. This research is related to evolutionary approaches presented previously, see [1]. However, this is the first time it has been applied to a MIP solver directly.

SCIP is a software framework for solving constraint programming (CP), mixed integer programming (MIP), and satisfiability (SAT) by integrating the solving techniques for each via *constraint integer programming* (CIP) [2]. SCIP was developed at the Zuse-Institute in Berlin (ZIB). The source code is available freely for academic and non-commercial use at the SCIP website: `http://scip.zib.de`. SCIP is currently one of the fastest, if not the fastest, non-commercial mixed integer programming (MIP) solvers [3], [4]. For solving MIP, CP or SAT problems, SCIP uses a branch-and-bound approach. The branch and bound process is complemented by linear programming (LP) relaxations and cutting plane separators as they are used in MIP solving an as well by constraint domain propagation and conflict analysis as is used for solving CP and SAT problems [2]. For this type of study an available source code for a state-of-the-art solver is a prerequisite.

A specific type of a MIP and one of the most extensively studied combinatorial optimization problem is the Multidimensional 0/1 Knapsack Problem (MKP) [5]. Due to it's simplicity, and the fact it is well studied, makes it for an ideal candidate for our study. It can be loosely described as that one is given a set of n items, each with a profit $c_j > 0$, and m resources, each with a capacity $b_i > 0$ where each item j consumes an amount $a_{ij} \geq 0$ from resource i. The goal is then to choose a subset of items that gives the maximum sum of profits while not exceeding the capacity of any of the resources. The problem can be formalized as the mixed integer program:

$$z^* = \text{maximize} \quad \boldsymbol{c}^T \boldsymbol{x} \tag{1a}$$

$$\text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{1b}$$

$$\boldsymbol{x} \in \{0,1\}^n \tag{1c}$$

where $\boldsymbol{A}$ is an $m \times n$ non-negative matrix, $\boldsymbol{x}$ is the variable vector of dimension n, and $\boldsymbol{b}$ and $\boldsymbol{c}$ are non-negative vectors of dimension m and n respectively. Each element x_j of the vector $\boldsymbol{x}$ is a $0-1$ decision variable indicating whether item j should be selected.

The paper is organized as follows. In Section II an overview of the branch and bound procedure is given and a detailed description of branching rules. Then in Section III we give the details of the evolutionary algorithm implemented. Since the evaluation of fitness functions will be very costly, we have chosen the computationally efficient (1+1)CMA-ES algorithm with a Cholesky factor update [6]. In Section IV the experimental study is presented and followed by its setup in Section IV and results of Section IV. A summary of main findings is then presented in conclusion. Notation for MIP-related theory closely follows that of [7].

II. BRANCH AND BOUND AND BRANCHING RULES

The *branch-and-bound* algorithm iteratively divides each MIP problem into subproblems, each having smaller solution space than the parent problem. At each step the LP-relaxation of each subproblem is solved where the MIP requirements of integrality have been cancelled. The MIP is NP-hard while the LP is solvable in polynomial time. After solving the LP, the problem is usually split into two parts by applying two disjoint bounds on a variable. For each subproblem, the new bounds are determined by using the upwards and downwards

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.204

rounded LP relaxation values, that is for variable x_j with LP solution of $\hat{x}_j$, the new bounds are $x_j \leq \lfloor \hat{x}_j \rfloor$ and $x_j \geq \lceil \hat{x}_j \rceil$, or in the case of binary problems $x_j = 0$ and $x_j = 1$.

During the solution process the algorithm creates a tree structure where each node represents a subproblem while the root of the tree corresponds to the initial problem instance. The *leaves* of the tree are problems that either have been solved or have yet to be solved. Unsolved leaf nodes are stored in a priority queue to be processed later. Leaf subproblems are considered solved if one of three criteria applies: 1) The subproblem is infeasible, 2) A feasible optimal solution has been found for the subproblem, 3) It has been proven that an optimal solution of the subproblem can not be any better than the best known feasible solution (the global upper bound) of the original problem. After a leaf node has been solved that particular subproblem is not divided any further and the corresponding node is said to have been *pruned*.

Other important components of the branch-and-bound based MIP solver are *node selection rules* that decide on what subproblem should be processed next after processing of each node [8], *presolving/propagation* [9], [10] that transform the problem into an equivalent one that should be easier to solve, *cut seperation* [11], [12] which finds a linear inequality that separates the current LP-optimum from the feasible integer domain and *primal heuristics* [8], [13] that try to find good feasible (incumbent) solutions without spending to much computing time while working outside the branch and bound framework.

One of the key factors in any branch-and-bound procedure is the decision of choosing which variable to select as the next branching variable. The set of potential branching variables are integer variables that are non-integer in the LP relaxation. Choosing the best candidate may significantly enhance the performance of the branch-and-bound procedure. This requires some sort of scoring function $\text{score}(\amalg) \in \mathbb{R}$ based on a set $\amalg$ of given measurements of quality. Let q_j^+ and q_j^- be some measures of quality for branching up and down respectively on a variable x_j. A possible *score function* could be defined as in [7] as

$$\text{score}(q_j^+, q_j^-) = (1-\mu) \cdot \min\{q_j^+, q_j^-\} + \mu \cdot \max\{q_j^+, q_j^-\} \tag{2}$$

where the score factor μ is some number between 0 and 1. However SCIP uses as a default a different score function [7]

$$\text{score}(q_j^+, q_j^-) = \max\{q_j^-, \epsilon\} \cdot \max\{q_j^+, \epsilon\} \tag{3}$$

where $\epsilon = 10^{-6}$. For each of these scoring functions the variable with the highest score is chosen as the variable to branch on, i.e. index of the next branching variable selected is $j = \arg\max_{k \in F}\{s_k\}$. Here $s_k = \text{score}(q_k^+, q_k^-)$ and F is the set of branching variables.

The *pseudocost branching rule* [14] estimates the quality of branching on a variable from past gains in objective function obtained by branching on the variable in question. In SCIP this rule is implemented as follows [7]: Let λ_j^- and λ_j^+ be the gains in objective value (Δ) per unit change in variable x_j at a particular node after branching in the corresponding direction or

$$\lambda_j^- = \frac{\Delta_j^-}{f_j^-}, \qquad \lambda_j^+ = \frac{\Delta_j^+}{f_j^+} \tag{4}$$

where $f_j^- = \hat{x}_j - \lfloor \hat{x}_j \rfloor$ and $f_j^+ = \lceil \hat{x}_j \rceil - \hat{x}_j$ are the fractional values of the LP solutions of the branched to subproblems. The pseudocosts of variable x_j are the arithmetic means

$$\Psi_j^- = \frac{\sigma_j^-}{\eta_j^-}, \qquad \Psi_j^+ = \frac{\sigma_j^+}{\eta_j^+} \tag{5}$$

Here σ_j^- is the sum of λ_j^- over all problems where the variable x_j was branched on downwards and where the LP relaxation was solved and was feasible. η_j^- is the number of these problems. σ_j^+ and η_j^+ are analogously defined for cases where the variable x_j was branched on upwards. Then the score function value for variable x_j, is

$$\text{psc}_s(j) = \text{score}(f_j^- \Psi_j^-, f_j^+ \Psi_j^+) \tag{6}$$

With *strong branching* [15], [16] the idea is to test for progress in the dual bound for each of the candidate variables before selecting the best one. The potential LP subproblems are solved to get estimates of the objective function gains. Strong branching is computationally expensive so usually some restrictions are set on either the number of variables tested and/or on the number of simplex iterations performed while solving the LPs.

Combining the two rules, *reliability branching* [2] uses strong branching estimates for *unreliable* pseudocosts. The pseudocost of a variable x_j is said to be unreliable if $\min\{\eta_j^+, \eta_j^-\} < \eta_{\text{rel}}$ where η_{rel} is the so called reliability parameter.

As opposed to past gains in the bound due to variable selection, an alternative measure of branching quality is the number of domain deductions on other variables made after branching on the variable in question. Analogous to pseudocosts the *inference branching* [2] value of a variable x_j, $j \in I$, is defined as

$$\Phi_j^- = \frac{\varphi_j^-}{\nu_j^-}, \qquad \Phi_j^+ = \frac{\varphi_j^+}{\nu_j^+} \tag{7}$$

with score function

$$\text{inf}_s(j) = \text{score}(f_j^- \Phi_j^-, f_j^+ \Phi_j^+) \tag{8}$$

where φ_j^-, φ_j^+ are the total number of inferences deduced after branching in the corresponding direction on the variable x_j. ν_j^- and ν_j^+ are weighted counts of corresponding subproblems where domain propagation has been applied. The idea is that a variable with a large historic inference value will be likely to produce more domain propagations and thus smaller subproblems in the future.

SAT solvers learn so called *conflict clauses* from the analysis of infeasible subproblems. A branching rule based on this approach is *conflict branching* [17] which takes into account whether corresponding candidate variables have been

used in recent conflict graph analysis to produce conflict constraints. The appearence of each variable in a clause is counted while periodically the counted sum is divided by a constant. A similiar rule, *conflict length branching*, uses the average lengths of the conflict clauses a variable appears in [17].

Lastly, *cutoff branching* favours branching variables in relation to the average incidences, where branching on the variable in question has led to either infeasible subproblems or to nodes pruned by bound, that is the number of *cutoffs* [17].

Similarly as in [17], but more generally, we define a *hybrid branching rule* as a branching rule whose score function is a linear combination of scaled versions of various quality measures and their scores according to eq. (3). Other components not traditionally used as quality measures in SCIP can also be integrated into the rule. We will refer to these components of the hybrid rule as *features*. The score function output for variable j would then be:

$$\begin{aligned} s_{hy}(j) = \omega_1 g_1\left(\frac{z_1(j)}{\tilde{z}_1}\right) + \omega_2 g_2\left(\frac{z_2(j)}{\tilde{z}_2}\right) \\ + \omega_3 g_3\left(\frac{z_3(j)}{\tilde{z}_3}\right) + \dots \end{aligned} \tag{9}$$

where $z_k(j)$ is the k-th component, or feature, of the rule and ω_k its assigned *weight*. $\tilde{z}_k$ is an appropriate normalization factor, often the feature mean over all j, and $g_k(\tau)$ is an appropriate scaling function.

The default branching rule in SCIP is a hybrid branching rule. It combines scores obtained from the score function of five different branching rules. As well as combining *reliability branching* and *inference branching*, SCIP's *hybrid reliability/inference branching* rule also weighs in scores from conflict branching, conflict length branching, as well as cutoff branching [17]. This type of hybrid branching rule is the current state-of-the-art [13]. For this reason its seems reasonable that new hybrid branching rules, customized for solving a class of instances such as multidimensional knapsack problems, could potentially be found whith an application of a direct search algorithm. Such an algorithm, and the one chosen for this study, is described in the following section.

III. (1+1)CMA-ES

The (1+1) covariance matrix adaptation evolutionary strategy is a single parent search strategy [6]. The parent solutions $\boldsymbol{g}$ are replicated (imperfectly) to produce the next generation $\boldsymbol{h} \leftarrow \boldsymbol{g} + \sigma\, \mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$. Here σ is a global step size and $\boldsymbol{C}$ is the covariance matrix of the d-dimensional zero mean Gaussian distribution. The replication is implemented in three basic steps:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{10a}$$

$$\boldsymbol{s} \leftarrow \boldsymbol{D}\boldsymbol{z} \tag{10b}$$

$$\boldsymbol{h} \leftarrow \boldsymbol{g} + \sigma \boldsymbol{s} \tag{10c}$$

where the covariance matrix has been decomposed into Cholesky factors $\boldsymbol{D}\boldsymbol{D}^\intercal$. The normally distributed random vector $\boldsymbol{z}$ is sampled from the standard normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The success probability of this replication is updated by

$$p_{succ} \leftarrow (1-\eta) p_{succ} + \eta\; I_{true}(\upsilon(\boldsymbol{h}) \leq \upsilon(\boldsymbol{g})) \tag{11}$$

where $\upsilon(\cdot)$ if the objective function or the *fitness*, that needs to be minimized. Here $I_{true}(\cdot)$ is the indicator function and takes the value one if its argument is true otherwise zero. The parameter η is the learning rate ($0 < \eta \leq 1$) and is set to $\eta = 1/12$. The initial value of $p_{succ} = 2/11$ is also the target success probability $\check{p}_{succ}$. Following the evaluation of the success probability the global step size is updated by

$$\sigma \leftarrow \sigma \exp\left(\frac{p_{succ} - \check{p}_{succ}}{\delta(1-\check{p}_{succ})}\right) \tag{12}$$

where $\delta = 1 + d/2$. The initial global step size will be problem dependent but should cover the intended search space. The parameter settings used are suggested by [18].

If the replication was successful, that is if $\upsilon(\boldsymbol{h}) \leq \upsilon(\boldsymbol{g})$, then $\boldsymbol{h}$ will replace the parent search point $\boldsymbol{g}$. Futhermore, the Cholesky factors $\boldsymbol{D}$ will be updated. Initially $\boldsymbol{D}$ and $\boldsymbol{D}^{-1}$ are set to the identity matrix $\boldsymbol{I}$ and $\boldsymbol{s}$ is set to $\boldsymbol{0}$. For $c_{cov} = 2/(d^2+6)$ and the threshold probability $\bar{p} = 0,44$, the update of the covariance matrix is then as follows [18]:

1. If $p_{succ} < \bar{p}$ then set

$$\boldsymbol{s} \leftarrow (1-\delta_{-1})\boldsymbol{s} + \sqrt{\delta^{-1}(2-\delta^{-1})}\boldsymbol{D}\boldsymbol{z} \tag{13a}$$

$$\alpha \leftarrow (1 - c_{cov}) \tag{13b}$$

Else set

$$\boldsymbol{s} \leftarrow (1-\delta^{-1})\boldsymbol{s} \tag{14a}$$

$$\alpha \leftarrow 1 - c_{cov}^2 \delta^{-1}(2-\delta^{-1}) \tag{14b}$$

2. Compute

$$\gamma \leftarrow \boldsymbol{D}^{-1}\boldsymbol{s} \tag{15a}$$

$$\beta \leftarrow \sqrt{1 + c_{cov}\|\gamma\|^2/\alpha} \tag{15b}$$

3. Compute

$$\boldsymbol{D} \leftarrow \sqrt{\alpha}\boldsymbol{D} + \sqrt{\alpha}(\beta-1)\boldsymbol{s}\gamma^\intercal/\|\gamma\|^2 \tag{16}$$

4. Compute

$$\boldsymbol{D}^{-1} \leftarrow \frac{1}{\sqrt{\alpha}}\boldsymbol{D}^{-1} - \frac{1}{\sqrt{\alpha}\|\gamma\|^2}(1-1/\beta)\gamma[\gamma^\intercal \boldsymbol{D}^{-1}] \tag{17}$$

The vector $\boldsymbol{s}$ of eq. 13a and 13a is already computed in eq. 10b. It has been shown that $\boldsymbol{D}^{-1}$ requires $\Theta(d^2)$ time, whereas a factorization of the covariance matrix requires $\Theta(d^3)$ time [18]. The Cholesky version of the (1+1)CMA is therefore computationally more efficient. The parameter values above are the same as in [18]. The maximum number of iterations, $k_{\max}$ was set at 1000.

The search algorithm described will now be implemented in an experimental study in order to discover new branching rules for the branch-and-bound procedure.

IV. Experimental Study

Multiple 0/1 knapsack instances were generated with similar methods as in [5]. All generated problem instances have the form of

$$z^* = \text{maximize} \quad \boldsymbol{c}^T\boldsymbol{x} \tag{18a}$$
$$\text{s.t.} \ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{18b}$$
$$\boldsymbol{x} \in \{0,1\}^n, \tag{18c}$$
$$\boldsymbol{c} \in \mathbb{R}^n_{\geq 0}, \ \boldsymbol{b} \in \mathbb{N}^m_0, \ \boldsymbol{A} \in \mathbb{N}^{m\times n}_0 \tag{18d}$$

The coefficients were randomly generated as

$$a_{ij} \sim \mathcal{U}\{0, a_{\max}\} \tag{19a}$$
$$b_i = \lfloor \tau \sum_{j=1}^{n} a_{ij} \rfloor \tag{19b}$$
$$c_j = \sum_{i=1}^{m} a_{ij}/m + \rho u_j a_{\max} \tag{19c}$$

Here $\mathcal{U}$ is the uniform distribution and τ is the so called tightness ratio. Along with the uniformly (and continuously) distributed parameter $u_j \sim \mathcal{U}(0,1)$, ρ sets the degree of correlation between the objective coefficients c_i and the constraint coefficients a_{1j} through a_{mj}.

The implementation of the (1+1)CMA-ES algorithm for this study can be described in the basic steps below:

1) Select a set of features or feature map $\boldsymbol{\phi} : \hat{x}_j \to \mathbb{R}^d$.
2) Initialize: $k \leftarrow 0$, $\boldsymbol{D}, \boldsymbol{D}^{-1} \leftarrow \boldsymbol{I}$, $\sigma \leftarrow 0.1$ and $\boldsymbol{s}, \boldsymbol{\omega} \leftarrow \boldsymbol{0}$.
3) Estimate a new weight vector from eq. 10 with eq. 10c becoming $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega} + \sigma\boldsymbol{s}$.
4) Evaluate fitness υ of $\boldsymbol{\omega}'$ by solving all problems in training set $\mathcal{P}_{\text{tr}}$ using branching score function $s_j = \boldsymbol{\omega}'^T\boldsymbol{\phi}(\hat{x}_j)$. Set $k \leftarrow k+1$. Update p_{succ} and σ.
5) If $\upsilon(\boldsymbol{\omega}') \leq \upsilon(\boldsymbol{\omega})$, then set $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}'$ and update $\boldsymbol{D}$ and $\boldsymbol{D}^{-1}$.
6) If $k < k_{\max}$ and $\sigma > \sigma_{\min}$ repeat from step 2. Else return $\boldsymbol{\omega}$.

For this implementation of the (1+1)CMA-ES algorithm, the fitness value υ is calculated in each iteration as the sum of all performances for a particular performance measure y_υ over all training instances in $\mathcal{P}_{\text{tr}}$. The optimization can the be formally defined as:

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \upsilon(\boldsymbol{\omega}) = \sum_{p\in\mathcal{P}_{\text{tr}}} y_\upsilon(\boldsymbol{\omega}^T\boldsymbol{\phi}, p) \tag{20}$$

with $y_\upsilon(\boldsymbol{\omega}^T\boldsymbol{\phi}, p)$ being the performance associated with using score function $\boldsymbol{\omega}^T\boldsymbol{\phi}$ when solving instance p. The computation time needed then for creating (training) of a branching rule is $t_{\text{tr}} = \bar{t} \times |\mathcal{P}_{\text{tr}}| \times k^*$, where $\bar{t}$ is the average runtime over all problems in $\mathcal{P}_{\text{tr}}$ over all iterations and k^* is the total number of iterations performed by the evolutionary algorithm over training set $\mathcal{P}_{\text{tr}}$.

As with regard to the performance of the evolved branching rules, the ideal number of training problems $|\mathcal{P}_{\text{tr}}|$ was unknown. Also unknown were the ideal combination as well as the number of features comprising the score function of the evolved branching rules. Therefore for the purposes of this experiment, these components were varied and multiple rules evolved for each combination of these components. The number of problems used for training was $|\mathcal{P}_{\text{tr}}| \in \{1, 5, 10\} \times 10^3$.

The features used as building blocks for the evolved rules used in this study are:

- $\text{rel}_{(\cdot)}$ - Reliability branching features
- $\text{psc}_{(\cdot)}$ - Pseudocost branching features
- con_s - Conflict branching score
- conl_s - Conflict length branching score
- inf_s - Inference branching score
- cut_s - Cutoff branching score
- fra - Fractionality or $\min\{f_j^-, f_j^+\}$
- ncon - Fraction of problem constraints the variable x_j appears in
- eff - Item efficiency measure as suggested by [19]

Used as features, from reliability and pseudocost branching rules as described in Section II, were the scores and the up/down quality measures q_j^+ and q_j^- as well as their minima, $\min\{q_j^+, q_j^-\}$, and maxima, $\max\{q_j^+, q_j^-\}$. Also used were the scores of the conflict, conflict length, inference and cutoff branching rules, as is the case with the SCIP hybrid/inference branching rule described in Section II. Further features are fractionality, the fraction of problem constraints a variable appears in, and item efficiency measure [19]. The last three features were deemed worthy to be included in the experiment since they might be especially valid for the knapsack problems.

Since the training problems are very small, runtime was not considered a reliable measure of performance for training purposes. The goal was then to minimize the total number of nodes Q and/or the total number of simplex iterations ς needed to solve the problems which are known to be highly correlated with runtime. Both performance measures were used as fitness measures for rule evolution while multiple rules were evolved in both cases. To increase the chance of discovering a good branching rule, as many branching rules as possible were created. By fully utilizing the available resources of computing power a total of $N_r = 146$ rules were created with the evolutionary algorithm. The feature maps (or in effect, branching rules) used in this study were of four types, denoted as EB_1, EB_2, EB_3 and EB_4, with each type composed of a unique set of features (Table I). Rules of type EB_1 have the same features as the SCIP's default hybrid inference/reliability branching rule. Shown are also the default weight values for the HIB rule. The low weight values for three of it's features, when compared to the reliability branching score weight value, are indicative of these features being mainly used for tie-breaking. While pseudocost measures are hybridized with strong branching measures in reliability branching, as described in Section II, the pseudocost branching quantities used in rules of types EB_2 and EB_3 are "pure" pseudocosts (without any strong branching starting), as are used in the pseudocost branching

Table I
TYPES OF RULES USED AND THEIR FEATURE BUILDING BLOCKS

Rule	$\text{rel}_{(\cdot)}$			$\text{psc}_{(\cdot)}$		con_s
	+/-	m/m	s	+/-	m/m	
HIB			1			10^{-4}
EB_1			✓			✓
EB_2	✓	✓	✓			
EB_3	✓	✓	✓	✓	✓	✓
EB_4	✓	✓	✓			

	conl_s	inf_s	cut_s	fra	ncon	eff
HIB	0	10^{-4}	10^{-2}			
EB_1	✓	✓	✓			
EB_2				✓	✓	✓
EB_3	✓	✓	✓	✓	✓	✓
EB_4						

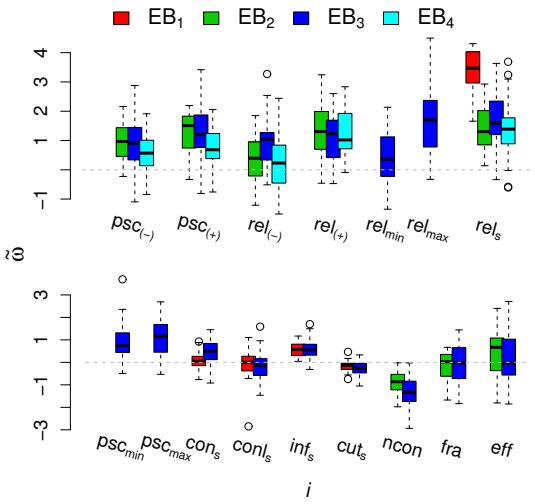

Figure 1. Distribution of evolved weight values returned for each feature by type of rule

rule described in the same section. The feature make-up of the experimental rules was chosen relatively ad hoc. The distribution of weight values of the N_r rules returned by the ES algorithm is summarized by the box plot of Fig. 1. For feature i, the weight value of component ω_i of weight vector $\boldsymbol{\omega}$ is normalized according to

$$\tilde{\omega}_i = \frac{\dim \boldsymbol{\omega}}{|\boldsymbol{\omega}|_1}\omega_i \tag{21}$$

with $|\boldsymbol{\omega}|_1$ being the 1-norm of the vector $\boldsymbol{\omega}$. Positive weight values are associated with features whose increase in value is associated with decrease in fitness value (better performance).

The ncon feature, which is a part of rules EB_2 and EB_3, has negative weight values for most of the evolved rules of that type. This indicates that variables involved in fewer constraints are, on average, better choices than others, at least while solving the instances of the training set. Negative weight values are more prominent for the cutoff branching score and the conflict length score, especially when used in the EB_3 rule. The fractionality, efficiency and conflict length score features seem to be the least effective when it comes to minimizing the fitness value, having the corresponding weights distributed around zero. The weights relating to the $\text{psc}(\cdot)$ and $\text{rel}(\cdot)$ features seem to have the largest impact on the fitness value. The weights of these pseudocost and strong branching related features are mostly in the positive, as would have been expected. For both pseudocost as well as the reliability branching features, the upwards measure appears more effective than the downwards measure and so do the maximums when compared to the corresponding minimums. These differences are more pronounced for the reliability branching features. Except for the reliability branching score, all of the score features used by the HIB rule have low apparent effectiveness which coincides with their tie-breaking status in the HIB rule.

Table II
SETS OF PARAMETERS USED FOR GENERATION OF TESTING INSTANCES

Parameter set	Parameters
n_{30}	$n \in \{30, 32, 34, ..., 48\}$
n_{40}	$n \in \{40, 42, 44, ..., 58\}$
n_{90}	$n \in \{90, 92, 94, ..., 108\}$
n_{100}	$n \in \{100, 250, 500\}$
m_5	$m \in \{5, 6, 7, 8\}$
m_7	$m \in \{7, 8, 9, 10\}$
m_{30}	$m \in \{5, 10, 15, 20, 25, 30\}$
τ_{65}	$\tau \in \{65, 70, 75, 80, 85\}$
τ_{25}	$\tau \in \{25, 50, 75\}$
ρ_{40}	$\rho \in \{0.40, 0.45, 0.50, 0.55, 0.60\}$

V. EXPERIMENTAL SETUP

For analysis of the performance of the branching rules created during this study a number of test instances were generated and solved by applying these rules with the SCIP solver. Some of the instances used for testing purposes are generated using the same instance generation parameters as the instances used for training. Further instances were generated by using different choice of generation parameters. These were then used for evaluation of rule performance over more general classes of instances. The testing instances were split into problem sets according to which sets of combinations (Table II) of the parameters n, m, τ, ρ and a_{max} were used to generate them (Table III). Table III shows which sets of parameters were used when generating each problem set. $|\mathcal{P}_{\text{te}}|$ is the number of test problems generated for each problem set. The first four test sets are for evaluation of the generalization abilities of the evolved rules with respect to the performance measures used as fitness variables, i.e. nodes and iterations. The fifth class of instances have similar values and combination of parameters as the instances used in [5]. The results for test set five are intended to show if using the evolved rules for solving instances much larger than the instances used for training will have the effect of

Table III
COMBINATIONS OF PARAMETER SETS USED FOR GENERATION OF TRAINING AND TESTING INSTANCES

Pr. set	$\lvert\mathcal{P}_{\text{te}}\rvert$	n	m	τ	ρ	a_{max}
1	1000	n_{30}	m_5	τ_{65}	ρ_{40}	9
2	1000	n_{30}	m_5	τ_{65}	ρ_{40}	999
3*	1000	n_{40}	m_5	τ_{65}	ρ_{40}	9
4	1000	n_{90}	m_7	τ_{65}	ρ_{40}	9
5	540	n_{100}	m_{30}	τ_{25}	.50	9
$tr^{\dagger}$	-	n_{30}	m_5	τ_{65}	ρ_{40}	9

*Unbounded integer knapsack instances
†Training instances

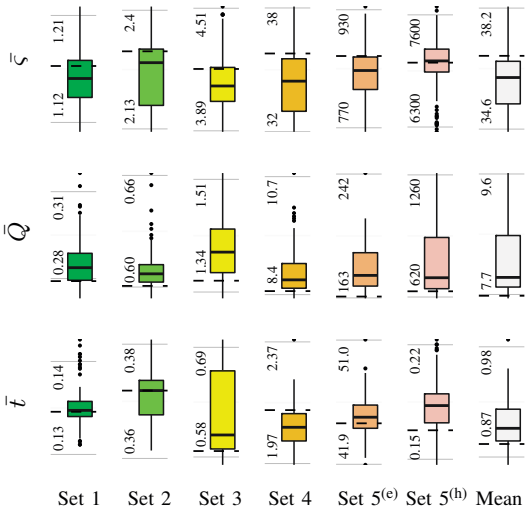

Figure 2. Distribution of average performances of all rules by problem set as well as the geometric mean ("Mean") over all sets. Averages for the HIB rule are shown for comparison (dashed lines).

decreasing runtime over the large instances. Each evolved rule was tested by solving all of the problems of the test sets with time limit set at $t = 500$ seconds. During testing all SCIP parameters, except of course those parameters which govern the choice of branching rule, were set at their default values.

VI. EXPERIMENTAL RESULTS

Fig. 2 shows the distribution of results for all rules over each of the test sets. Shown separately are distribution of average number of nodes $\bar{Q}$, iterations $\bar{\varsigma}$, as well as runtime $\bar{t}$. Test set 5 is split into easy (e) and hard (h) instances. For the hard instances of set 5 the average relative gap $\bar{\%}_{\text{LP}}$ in percentages is used instead of the runtime, that is the relative gap of obtained solution values z, to the optimal objective value z_{LP} of the LP-relaxation ($\%_{\text{LP}} = (z_{\text{LP}} - z)/z_{\text{LP}}$). Except for the tough instances of set 5, the majority of the evolved rules seem to perform better than the HIB rules over all test sets when considering number of iterations performed. The reverse is true for number of nodes performed, the evolved rules generally did worse than the HIB rule. When

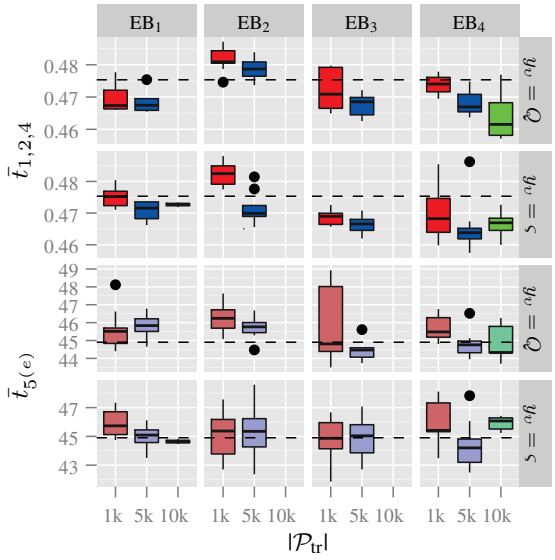

Figure 3. Distribution of the geometric mean of average runtimes over test sets 1,2 and 4 (top two rows) and average runtimes over the easy instances of test set 5 (bottom two rows), by rule type and type of fitness variable y_v. Averages for the HIB rule are shown for comparison (dashed lines).

considering runtime, test set 4 is the only set where majority of the rules did better than the HIB rule while about half the rules did better over test sets 1 and 2. Looking at the distributions of the geometric means shown in the rightmost boxplots, it is clear that a majority of the rules perform better than the HIB rule, when considering iterations. However, a majority of the evolved rules perform less well than the HIB rule over when considering both the number of nodes as well as runtime.

Fig. 3 shows distribution of the geometric mean of runtimes averaged over sets 1,2 and 4 and the average runtimes over the easy instances of set 5 respectively. Since the distributions of runtimes vary a great deal between test sets, the geometric mean is used rather than the arithmetic one. The distributions shown are according to rule type and type of fitness variable y_v used by the ES algorithm. For most of the boxplots shown, the rules trained with $|\mathcal{P}_{\text{tr}}| = 5000$ instances perform better than rules trained with $|\mathcal{P}_{\text{tr}}| = 1000$ instances. Roughly speaking, the rules of type EB_4 trained with $|\mathcal{P}_{\text{tr}}| = 10000$ instances do better than their counterparts trained with $|\mathcal{P}_{\text{tr}}| = 5000$ instances if the fitness variable was nodes while the opposite is true if the fitness variable was iterations. There were to few rules of type EB_1 trained with $|\mathcal{P}_{\text{tr}}| = 10000$ instances for making any conclusions about their performance. For the easy instances of test set 5, rules trained with both types of fitness variable y_v have shorter runtimes than the HIB rule.

Table IV shows the average number of nodes generated for each test set for 5 individual rules. Shown are results for the rules that had the best performance among all rules, averaged over test sets 1-4, when comparing total number of nodes

Table IV
AVERAGE NUMBER OF NODES IN THOUSANDS (TIME IN SECONDS) PRODUCED BY THE 5 RULES WITH THE FEWEST NODES PRODUCED ON AVERAGE

			$\bar{Q}$(thousand)				
Rule	$\lvert\mathcal{P}_{\text{tr}}\rvert$	y_v	Set 1	Set 2	Set 3	Set 4	Set 5 (easy)
HIB	-		0.279	0.601	1.36	8.47	164 (44.9)
RB	-		0.280	0.605	1.35	8.57	167 (45.3)
EB_4	5k	Q	**0.278**	**0.594**	**1.33**	**8.35**	171 (44.9)
EB_3	5k	Q	**0.274**	**0.596**	**1.35**	**8.37**	171 (**43.7**)
EB_1	1k	Q	**0.278**	0.603	**1.35**	**8.36**	170 (**44.8**)
EB_1	5k	Q	**0.278**	0.603	**1.35**	**8.35**	165 (**44.7**)
EB_1	5k	Q	**0.275**	0.601	1.36	**8.37**	166 (45.5)
$\lvert\mathcal{P}_{\text{te}}\rvert$			1000	1000	1000	1000	270

Table V
AVERAGE NUMBER OF SIMPLEX ITERATIONS IN THOUSANDS (TIME IN SECONDS) PERFORMED BY THE 5 RULES WITH THE FEWEST ITERATIONS PERFORMED ON AVERAGE

			$\bar{\varsigma}$(thousand)				
Rule	$\lvert\mathcal{P}_{\text{tr}}\rvert$	y_v	Set 1	Set 2	Set 3	Set 4	Set 5 (easy)
HIB	-		1.17	2.31	4.20	35.8	867 (44.9)
RB	-		1.17	2.31	4.22	36.3	868 (45.3)
EB_4	10k	ς	**1.14**	**2.13**	**4.00**	**32.3**	**817** (45.5)
EB_4	10k	ς	**1.11**	**2.12**	**3.95**	**32.6**	**820** (46.2)
EB_4	5k	ς	**1.14**	**2.16**	**3.88**	**32.4**	**830** (**44.8**)
EB_4	1k	ς	**1.13**	**2.17**	**3.97**	**32.3**	**811** (**44.3**)
EB_3	5k	ς	**1.12**	**2.14**	**4.02**	**32.3**	**823** (46.0)
$\lvert\mathcal{P}_{\text{te}}\rvert$			1000	1000	1000	1000	270

generated. Table V shows the analogous resulting number of iterations performed by the 5 best performing rules when comparing total number of iterations performed. Results are shown for all test sets except for the tougher instances of set 5 since for these instances time limit had an effect on the number of nodes/iterations generated/performed. For test set 5 the average runtimes in seconds are shown in parenthesis.

The order of the rules is in accordance of their average rank, with better performing rules above the less well performing rules. The values shown are each rule's average over all instances belonging to a particular set. Also shown are the performances of the hybrid inference/reliability branching rule (HIB) and the reliability branching rule (RB). For each rule considered, shown are the rule type, number of training problems $|\mathcal{P}_{\text{tr}}|$ iterated over while evolving the rule, and the type of fitness variable y_v used (Q indicating nodes and ς iterations). Shown in bold are results for rules that outperform both the HIB and RB rules. Over-lined numbers state significantly better performance than both the HIB and RB rules at the 95% significance level. Significance was tested with the Wilcoxon signed-rank test.

The highest ranking rules when comparing number of nodes have all been evolved using total number of nodes as a measure of fitness. Analogously, the highest ranking rules when comparing number of iterations have all been evolved using iterations as a measure of fitness. With the exception of test set 1, the performances according to number of iterations of the rules of Table V are more often significantly better than the SCIP rules when compared to the performances in nodes of Table IV, especially for test sets 2 and 5. For the results of rules of both tables over set 5, half of the rules show improvement in runtime when compared to the HIB and RB rules, all of them significantly so. The rules of Table IV do so despite all of them having inferior performance with respect to nodes generated. Table VI shows the average runtime for 10 individual rules over the instances of set 5 split into six subsets according to hardness. Shown are results for rules with the lowest ranks among all rules, averaged over test sets 1,2 and 4, when comparing average runtime. The

Table VI
AVERAGE RUNTIME IN SECONDS/GAP IN PERCENTAGES FOR 6 SUBSETS OF TEST SET 5. RESULTS ARE FOR THE 10 RULES WITH LOWEST AVERAGE RANKING, OVER SETS 1,2 AND 4, OF RUNTIME SPENT

			Set 5 (easy)[$\bar{t}$]			Set 5 (hard)[$\bar{\%}_{\text{LP}}$]		
Rule	$\lvert\mathcal{P}_{\text{tr}}\rvert$	y_v						
HIB	-	-	7.56	49.3	77.9	0.182	0.150	0.171
RB	-	-	7.34	48.1	80.6	0.182	0.155	0.165
EB_4	5k	ς	**6.93**	**45.5**	81.2	0.234	0.152	0.177
EB_4	10k	Q	**7.23**	48.1	**77.5**	0.186	0.157	0.176
EB_4	5k	ς	**6.99**	**45.9**	**74.6**	0.231	0.154	0.174
EB_4	10k	Q	7.41	**47.3**	**76.4**	0.194	**0.144**	0.177
EB_4	5k	ς	**7.19**	**45.1**	**77.2**	0.239	0.152	0.180
EB_4	10k	Q	**7.25**	48.1	**77.6**	0.188	0.157	0.171
EB_4	10k	ς	**6.76**	**47.9**	81.9	0.242	0.155	0.182
EB_4	10k	Q	7.46	**47.8**	**77.7**	0.190	0.151	0.175
EB_4	5k	ς	7.42	**47.1**	**75.6**	0.234	**0.149**	0.184
EB_4	5k	Q	**6.82**	50.2	**76.1**	0.184	0.153	0.176
$\lvert\mathcal{P}_{\text{te}}\rvert$			90	90	90	90	90	90

runtimes shown in Table VI are in general not significantly lower than the runtimes of the HIB and RB rule. The 10 rules perform relatively better over the easy instances than over the hard instances. Since rules of type EB_4 have on average the lowest runtimes for test sets 1,2 and 4, the results of Table VI happen to be all for rules of that type.

VII. SUMMARY AND CONCLUSIONS

As was shown in Fig. 2 the evolved rules were in general well suited to minimize the overall number of simplex iterations performed while solving the instances of all the test sets. The same can not be said for minimizing number of nodes generated however. There is evidence that 1000 instances are not sufficient for training an effective branching rule on, while it might be beneficial to train with more than 5000 instances. As well as being promising for set 5, the results for runtime are especially encouraging for test set 4. The set consists of instances that were generated with the same sets of parameters τ,ρ and a_{max} as the training problems while the values of n and m are larger. The runtimes for test set 1, which instances were generated with

the same sets of parameters as the training set, are on average not as good when compared to the HIB rule. Although the rules did better on average on set 4 with respect to iterations, it does not by it self explain the overall reduction in runtime. However, since the rules are optimized so as to minimize the sum, or average of the fitness values over the training instances, the rules might be better adapted to the hard instances of the training problems. Then solving test set of larger instances, generated with otherwise the same set of parameters as the training instances, might perhaps be a well suited task for these rules.

For this study to be deemed successful only a single rule that can consistently outperform the state-of-the-art HIB branching rule needs to be discovered. For future study however, it might be feasible to examine the overall performance of all rules in order to be better able to save resources later. Indeed, when looking at overall performance in runtime of the evolved rules, they are not particularly good at solving the integer instances of test set 3 or the harder problems of test set 5. Furthermore, the geometric mean of average runtimes over all test sets might indicate that the evolved rules outperform the comparison rules only by chance. However, as can be seen by the performance in nodes and iterations of Tables IV and V, some of the rules, as well as spending less runtime than the HIB and RB rules solving the easy instances of set 5, significantly outperform both the HIB and RB rules consistently over most of the other test sets, and more often than one would expect would happen by chance alone. As shown in these tables as well as in Table VI, the results for runtime spent solving the easy instances of set 5 are encouraging. The results for the gap percentage shown in Table VI are poorer. These might be an indication that the method of using (1+1)CMA-ES in this way, to use multiple small instances to train branching rules or other decision rules, to be better at solving instances of larger size, might not be sound. On the other hand it might be possible to train decision rules using the relative gap as a measure of fitness while solving small parts of larger instances. This could be investigated in a possible future study. Further study might also involve adapting (1+1)CMA-ES or another algorithm to be able to select which features to use, as well as optimizing the weights, so as to eliminate the need for choosing the features beforehand.

We have shown that it is possible to use the (1+1)CMA-ES algorithm for training of branching rules to be efficient at solving a special class of mixed integer programs, specifically multidimensional knapsack problems. We showed that it is possible to use as a fitness variable the number of nodes generated, as well as number of simplex iterations performed, to create branching that were faster than the HIB and RB rules solving the test instances. We further showed that the evolved rules generalize to a wider class of instances as well as those larger in size. It remains to be seen whether this method, or a modification of it, is practical for the creation of effective branching rules for solving of instances of even larger sizes.

REFERENCES

[1] E. K. Burke, M. Gendreau, M. Hyde, G. Ochoa, G. Kendall, E. Özcan, and R. Qu, "Hyper-heuristics : A Survey of the State of the Art," *Journal of the Operational Research Society*, 2013.

[2] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, Jan. 2009. [Online]. Available: http://link.springer.com/10.1007/s12532-008-0001-1

[3] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," Technische Universität Wien (Vienna University of Technology), Tech. Rep., 2012.

[4] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter, "Miplib 2010," *Mathematical Programming Computation*, vol. 3, no. 2, pp. 103–163, Jun. 2011. [Online]. Available: http://link.springer.com/10.1007/s12532-011-0025-9

[5] J. Puchinger, G. R. Raidl, and U. Pferschy, "The multidimensional knapsack problem: Structure and algorithms," *INFORMS Journal on Computing*, vol. 22, no. 2, pp. 250–265, 2010.

[6] C. Igel, T. Suttorp, and N. Hansen, "A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 453–460.

[7] T. Achterberg, "Constraint Integer Programming," Ph.D. dissertation, 2007.

[8] L. A. Wolsey, "Mixed integer programming," *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[9] T. J. Van Roy and L. A. Wolsey, "Solving mixed integer programming problems using automatic reformulation," *Operations Research*, vol. 35, no. 1, pp. 45–57, 1987.

[10] M. W. Savelsbergh, "Preprocessing and probing techniques for mixed integer programming problems," *ORSA Journal on Computing*, vol. 6, no. 4, pp. 445–454, 1994.

[11] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," 1958.

[12] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj, "Gomory cuts revisited," *Operations Research Letters*, vol. 19, no. 1, pp. 1–9, 1996.

[13] A. Lodi, "Mixed Integer Programming Computation," in *50 Years of Integer Programming 1958-2008*, M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. 16, pp. 619–645. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-68279-0

[14] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, "Experiments in mixed-integer linear programming," *Mathematical Programming*, vol. 1, no. 1, pp. 76–94, 1971.

[15] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Finding cuts in the tsp," 1995.

[16] J. T. Linderoth and M. W. P. Savelsbergh, "A Computational Study of Search Strategies for Mixed Integer Programming," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 173–187, Jan. 1999. [Online]. Available: http://joc.journal.informs.org/cgi/doi/10.1287/ijoc.11.2.173

[17] T. Achterberg and T. Berthold, "Hybrid Branching," pp. 309–311, 2009.

[18] T. Suttorp, N. Hansen, and C. Igel, "Efficient covariance matrix update for variable metric evolution strategies," *Machine Learning*, vol. 75, no. 2, pp. 167–197, Jan. 2009. [Online]. Available: http://link.springer.com/10.1007/s10994-009-5102-1

[19] A. Freville and G. Plateau, "An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem," *Discrete Applied Mathematics*, vol. 49, no. 1-3, pp. 189–212, Mar. 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X94902097