DIVERSITREE: COMPUTING DIVERSE SETS OF NEAR-OPTIMAL SOLUTIONS TO MIXED-INTEGER OPTIMIZATION PROBLEMS

Izuwa Ahanor, Hugh Medal
Department of Industrial and Systems Engineering, University of Tennessee-Knoxville
iahanor@vols.utk.edu, hmedal@utk.edu

Andrew C. Trapp
Data Science Program, Worcester Polytechnic Institute
WPI Business School, Worcester Polytechnic Institute
atrapp@wpi.edu

ABSTRACT
While most methods for solving mixed-integer optimization problems seek a single optimal solution, finding a diverse set of near-optimal solutions can often be more useful. State of the art methods for generating diverse near-optimal solutions usually take a two-phase approach, first finding a set of near-optimal solutions and then finding a diverse subset. In contrast, we present a method of finding a set of diverse solutions by emphasizing diversity within the search for near-optimal solutions. Specifically, within a branch-and-bound framework, we investigate parameterized node selection rules that explicitly consider diversity. Our results indicate that our approach significantly increases diversity of the final solution set. When compared with existing methods for finding diverse near-optimal sets, our method runs with similar run-time as regular node selection methods and gives a diversity improvement of up to 140%. In contrast, popular node selection rules such as best-first search gives an improvement of no more than 40%. Further, we find that our method is most effective when diversity is emphasized more in node selection when deeper in the tree and when the solution set has grown large enough.

Keywords near-optimal solutions · mixed-integer programming · diversity

1 Introduction
It is often useful to find sets of near-optimal solutions to optimization problems rather than a single solution. In particular, for these multiple near-optimal solutions to be usable, they should be diverse, to ensure that decision makers are presented with a variety of options.

One class of methods for generating diverse sets of near-optimal solutions to mixed-integer optimization (MIO) problems uses a two-phase approach. In the first solution generation phase, an oracle finds a set of near-optimal solutions without considering diversity. For example, [Danna et al.] (2007) developed the ONETREE oracle for this purpose. In the second diverse subset selection phase the output set from the first phase is processed by heuristics or an
optimization algorithm to select a small subset of the input set with maximum diversity. While this approach works well for problems with a small number of near-optimal solutions, for MIOs with a very large set of near-optimal solutions it is not practical to find the complete set. As a result, the first phase can only compute a subset of near-optimal solutions. If this subset is not diverse, then the smaller subset computed by the second phase may also lack diversity. Because existing approaches for the first-stage solution generation phase do not consider diversity, the near-optimal sets they generate often lack diversity (see §4). To address this problem, we developed a method - DIVERSITREE - to emphasize diversity in the solution generation phase. The output from our solution generation oracle can be used by itself or as a more diverse set input for diverse subset selection methods.

1.1 Motivation

In many applications it is desirable to obtain a small number of near-optimal solutions for analysts to consider in their decision-making process. There are a number of specific applications in which it is desirable to find multiple near-optimal solutions to optimization problems including the correct identification of metabolic activity of cells and tissues in metabolic networks (Rodríguez-Mier et al. (2021)), identifying alternate near-optimal structural designs (He et al. (2020), providing policies that are more robust to data changes in reinforcement learning and machine learning (Kumar et al. (2020), Eysenbach et al. (2018)), enabling exploration and mapping searches to broader but specific solutions in large search spaces by including near-optimal search results in search requests (Mouret and Clune (2015), Zahavy et al. (2021), adding more artistic alternatives to structural topology optimization (Cai et al. (2021), He et al. (2020), providing competitive alternatives to facility location and location routing problems (Church and Baez (2020), Schittekat and Sørensen (2009), diversifying software deployment to enable stronger computer software security (Tsoupidi et al. (2020), aiding motif finding in computational molecular biology (Zaslavsky and Singh (2006), generating multiple near-optimal group preferences in computational social choice analysis (Boehmer and Niedermeier (2021)), and broadening architectural testing in processor design (Van Hentenryck et al. (2009)).

Many of these applications have a large number of near-optimal solutions (see Tables 3). For instance, both the routing problem from (Ceria et al. (1998)) and the multi-period facility location problem studied in (Eckstein (1996)) are known to have more than 10,000 solutions with objective value within 1% of optimum. In problems with many near-optimal solutions, there is a need to identify a small subset of near-optimal solutions that is representative of the whole. One measure of how well a subset represents the whole is the diversity of the subset. Unfortunately, methods for finding near-optimal solutions such as ONETREE find near-optimal sets consisting of solutions that are not very diverse (see Example provided in §5). There are several important contexts in which it is useful to have a diverse set of near-optimal solutions. First, in some design problems decision makers seek a set of designs to choose from, which allows them to consider other difficult-to-model factors when selecting a single design (Joseph et al. (2015)). A similar context is in statistical model selection for a specific domain. Presenting a domain expert with a diverse set of models with similar fit allows the expert to select the model that best matches intuition. Several new study hypothesize that for many machine learning tasks a set of models exist with near-minimal loss (Semenova et al. (2019), also see §9 of Rudin et al. (2022). Second, using optimization for decision problems can often be an iterative process in which a MIP solution is presented to a decision-maker and then the decision-maker determines that the solution violates an important side-constraint that was not included in the model. If the decision-maker is only given a single solution, the model must be re-solved after adding the side-constraint. However, if a diverse set of near-optimal solutions is given, then the decision-maker may be able to find a good solution in the set that does not violate the side-constraint, avoiding the need to re-solve the model. Third, in contexts in which solutions are implemented repeatedly, it can be useful to alternate between a diverse set of near-optimal solutions. For example, one may use an MIP model to match workers to jobs in order to minimize total time, but then implement different near-optimal matchings each day in order to increase cross-training. Lastly, a set of near-optimal solutions can be used to measure and explain the importance of variables in applications such as statistical model selection (see §9 of Rudin et al. (2022)). If a feature is present in a large number of diverse near-optimal solutions, then that provides evidence that the feature is an important predictor of the response variable.
In two-phase approaches the output of the solution generation phase is the input to the diverse subset selection phase. Solution generation methods that do not include diversity in the solution generation phase would use this approach with the intent of collecting enough solutions to cover the solution space in the first phase and hence a diverse set in the second phase. Apart from the additional time this takes, if the solution set computed in the solution generation phase lacks diversity, then the subset computed by the diverse subset selection phase may also lack diversity. To address this gap, this research explores methods that encourage diversity during the solution generation phase.

1.2 Related Work

Although there is some work on finding near-optimal solutions to continuous optimization problems (see Lavine (2019)), most of the work has been in relation to mixed-integer optimization (MIO) problems.

There are a number of different algorithms for generating a set of near-optimal solutions to a MIO problem including the branch-and-count method developed by Achterberg et al. (2008) which is based on detecting "unrestricted subtrees" in the branching tree, a decision-diagram method from Serra (2020), the ONE TREE method (currently implemented in GAMS-CPLEX) given by Danna et al. (2007) which extends the branching tree used to solve the MIP problem to generate near-optimal solutions. In addition, approaches have been developed for specific classes of problems including using dynamic programming to generate multiple solutions to graph based problems (Baste et al. (2022)), methods to represent near-optimal solutions compactly (Serra and Hooker (2020)) and algorithms specific to topology optimization problems (Wang et al. (2018)). While these methods are effective at finding near-optimal solutions, they do not necessarily compute solution sets that are diverse. One of the few papers that discusses balancing diversity and optimality is Zhou et al. (2016). They developed a Dual Diverse Competitive Design (DDCD) method that formulates balancing optimality and diversity as an optimization problem which maximizes diversity, subject to constraints on the performance penalty. Their method however focuses on generating competitive designs that would give diverse solutions while the goal of this article is a method for general MIO problems.

There are a number of different approaches for finding a diverse set of near-optimal solutions to an optimization problem. In the sequential approach (Danna et al. (2007)), the optimization problem is solved multiple times, and after each solve a constraint is added that requires the next optimal solution be different from the previous. The variable copy approach adds \( k \) copies of the variables, one copy for each near-optimal solution desired, in an optimization model and adds constraints to enforce that the solutions are different (Cameron et al. (2021)). The authors of Greistorfer et al. (2008) compared the sequential and variable copy approaches for the problem of finding two diverse near-optimal solutions and found that the sequential approach usually required less computation time and yielded solutions that were nearly as good as the simultaneous approach. Further, population-based metaheuristics are a natural approach for finding diverse sets of solutions because these algorithms operate on a set of solutions. For instance, Glover et al. (2000) used a scatter-search algorithm to find a diverse set of solutions to MIPs. While metaheuristics are often effective in practice, their main drawback is their inability to guarantee that the entire set of near-optimal solutions has been found. Finally, two-phase approaches use an oracle such as the ONE TREE algorithm (Danna et al. (2007)) to find a set of near-optimal solutions (not necessarily diverse). Then the second phase inputs this set of near-optimal solutions and chooses a subset that maximizes diversity. Regarding the second phase, Danna and Woodruff (2007) developed exact and heuristics algorithms to find a diverse subset. The work by Glover et al. (1998) which proposed four different heuristics algorithms for generating the most diverse solutions from a larger set does not consider how close they are to the optimal objective. Kuo et al. (1993) proposed two different methods that use linear programming to select the most diverse solutions from a solution set. Schwind et al. (2020) developed methods for computing a small subset of solutions that represent the larger set using bi-objective optimization. The authors of Danna et al. (2007) found that the sequential and simultaneous methods do not scale as the number of solutions that are been generated increases. The approach developed in this article combines both phases of the two-phase approach, emphasizing diversity while searching for a near-optimal set.
1.3 Contributions and Findings

This article reports results of experiments that examine how using different node selection rules within a branch-and-bound framework affects the diversity of the computed near-optimal solution set. In addition, we also present experimental results for new node selection rules that prioritize solution diversity. Our specific contributions are the following:

1. **Diversity of solution sets computing by existing node selection rules.** We investigated the diversity of near-optimal sets generated using popular node selection rules within a branch-and-bound algorithm and found that the effect of node selection rules on solution set diversity is problem specific, the best first search rule typically gave better diversity solutions overall.

2. **A new node selection rule that emphasizes solution set diversity.** We investigated a new node selection rule used within a branch-and-bound framework that selects new nodes in the tree based on a tradeoff between the relaxation of the node as well as the node diversity score.

3. **Best parameters for diverse-emphasizing node selection rules.** We investigated several different parameterized node selection rules for finding diverse solution sets that include parameters that control factors such as the depth of a node within the branch-and-bound tree and the number of solutions currently in the solution set and found that the greatest diversity is obtained when waiting to emphasize diversity until a small number of solutions is gathered or when selecting solutions deeper in the tree. Clustering problems into groups based on the diversity results from the training set, we identified four groups of problems and corresponding parameter settings.

4. **Benefits of emphasizing diversity in node selection.** Using the four groups of parameter settings, we found that a branch-and-bound type algorithm generates solution sets that are up to 140% more diverse than an existing method for finding diverse near-optimal sets. Regarding runtime, we found that our method of emphasizing diversity within the node selection procedure runs in similar time as regular node selection methods that do not emphasize diversity and results in a better tradeoff between runtime and solution diversity than competing approaches. Importantly, we found that these four groups of parameter settings performed well even when applied to a “test set” of problems that were not used for parameter tuning.

1.4 Outline of the paper

The remainder of this article first gives background information in §2, followed by a description of the diversity-emphasizing node selection rules tested in this work (§3). Experimental results on the training and test set data is given in §4 and §5 presents an example of using DIVERSITree on real data before concluding in §6.

2 Background

This article is interested in finding diverse sets of near-optimal solution to the following problem.

\[ z^* = \min_{x \in X} c^T x \quad \text{where} \]

\[ X = \{ x \in \mathbb{R}^d : Ax \leq b, x_i \in \mathbb{Z}, \forall i \in I \subseteq \{1, \ldots, d\} \} \]

Let \( S_q = \{ x \in X : c^T x \leq (1 + q)z^* \} \) denote a set of \( q \%-\)optimal solutions to equation (1). In the case that \( S_q \) is small, it is sufficient to use a near-optimal solution generation algorithm such as ONETree (Danna et al. (2007)) to generate the complete set and present it to a decision maker. In the case in which \( S_q \) is large but not very large (i.e., 1,000 or less) one can use the following two-phase approach. In the first phase, use a solution generation algorithm to
obtain the complete set of near-optimal solutions \( S_q \). Then in the second phase, use a diverse subset selection algorithm (see Danna and Woodruff (2009)) to find a small subset of \( S_q \) of cardinality \( p \), solving the following problem:

\[
\max_{S \subseteq S_q, |S| = p} D(S)
\]  

(2)

where \( D(S) \) is a measure of the diversity of solution set \( S \). However, in many cases, the set \( S_q \) is very large (i.e., greater than 10,000). In particular, for about half of the instances tested in Danna and Woodruff (2009), \(|S_q| > 10,000\). For these instances, the authors limited the input to the second phase to a subset of \( S_q \) consisting of 10,000 solutions obtained by the ONEtree algorithm Danna et al. (2007). That is, the first phase finds a subset \( \bar{S} \subseteq S_q (|\bar{S}| = 10,000) \) without considering diversity and then the second phase solves:

\[
\max_{S \subseteq \bar{S}, |S| = p} D(S)
\]  

(3)

Thus, these problem instances with very large \( S_q \) present two issues:

- **Computational.** These problems could not be solved to optimality in the second-phase of the two-phase approach because of memory limitations or exceeding a 10-day time limit. As a result, heuristics had to be used. For smaller problems these heuristics appear to produce solutions that nearly maximize diversity. However, it is not known whether the heuristics produce good solutions to the large problems.

- **Solution quality.** Using the subset \( \bar{S} \) as an input to the second phase rather than the complete set \( S_q \) could result in a loss of diversity if the input subset is not diverse. This loss of diversity could occur either with heuristic or exact methods.

To address these issues, we seek to solve equation (3) directly rather than using a two-phase approach. Specifically, we examine how to modify the exploration strategy of the solution generation phase in order to increase the diversity of the subset \( \bar{S} \).

### 2.1 Computing near-optimal sets

Given a mixed integer programming problem, we assume that it is possible to find an oracle capable of enumerating a set of all or a sufficiently high number of feasible near-optimal solutions for the problem. We use the branch-and-count method as the oracle because it shows significant speed at enumerating near-optimal solutions by detecting and pruning infeasible subtrees (Achterberg et al. (2008)). The branch-and-count method is also implemented in SCIP (Gamrath et al. (2020)) giving us access to the branching tree and implementation methods of the oracle to modify as we require.

The branch-and-count algorithm is an extension of the branch-and-cut algorithm (a variant of branch-and-bound that adds valid inequalities to the formulation during the traversal of the tree in order to strengthen the LP relaxation) and runs by going through the entire search tree generated during the regular branch-and-cut process and collects all feasible solutions at each node if that node is detected to represent an unrestricted subtree. The authors prove that the subtree at a node in the search tree is unrestricted if all constraints at that subtree are satisfied by all possible variable assignments of values in the subtree’s domain i.e., the constraints are locally redundant at that node. This way, they can construct solution vectors for that subtree and avoid traversing every node in that subtree. This significantly reduces the number of nodes this algorithm visits. The pseudocode is given in algorithm 1.

To use this algorithm to enumerate the set \( \bar{S} \), we first solve equation (1) to find an optimal objective value \( z^* \). Next, we pass the number of near-optimal solutions requested (\( p_1 \)) as input to the algorithm and add the constraint \( c^T x \leq (1 + q)z^* \) to equation (1) and solve using the branch-and-count algorithm.
Algorithm 1 Branch and Count Pseudocode

1: INPUT $p_1$, MIP
2: ADD constraint $c^T x \leq (1 + q) z^*$ to the initial MIP problem
3: ADD the initial MIP problem to the queue of active nodes $Q$
4: $\bar{S} \leftarrow \emptyset$
5: while $Q$ is not empty and $|\bar{S}| < p_1$ do
6: use node selection rule to dequeue node $i'$ from list $Q$
7: if node $i'$ is UNRESTRICTED then
8: Add solution to $\bar{S}$
9: else if node $i'$ is INFEASIBLE then
10: prune
11: end if
12: end while
13: return $\bar{S}$

2.1.1 Node Selection Rules

During line 6 of the branch-and-count search, the node selection rule (see line 6 of Algorithm 1) decides which node from the queue of active nodes is selected as the next node. Popular node selection rules include:

- **Best First Search (BestFS)** - selects the node with the best bound, i.e., for minimization problems, select $i'$ as the next node according to

  $$i' \in \arg \min_{i \in O} \{LB_i\},$$

  where $LB_i$ is the lower bound for node $i$.

- **Depth First Search (DFS)** - nodes encountered as the branch and bound search tree is traversed are added to a queue and are selected in Last In First Out (LIFO) order.

- **Breadth First Search (BrFS)** - nodes are added to a queue and processed using the First In First Out (FIFO) order.

- **Upper Confidence bounds for Trees (UCT) (Gamrath et al. (2020))** - selects the next node $i'$ as a node with the best UCT_score, i.e.

  $$i' \in \arg \min_{i \in O} \{UCT\_Score_i\},$$

  $UCT\_score$ is calculated as:

  $$UCT\_Score_i = LB_i + \rho \frac{V_i}{v_i}$$

  where $v_i$ and $V_i$ are the number of times the algorithm has visited node $i$ and its parent, respectively. $\rho$ is a weight parameter chosen by the user.

- **Hybrid Estimate (HE) (Gamrath et al. (2020))** - selects the next node $i'$ as a node having the best HE_Score, i.e.:

  $$i' \in \arg \min_{i \in O} \{HE\_Score_i\},$$

  $HE\_Score$ is calculated as:
DiversiTee

\[ HE_{\text{Score}}_i = (1 - \rho) \hat{L}B_i + \rho \hat{L}\hat{B}_i \]

\( \hat{L}B_i \) is the estimated value of the best feasible solution in subtree of node \( i \) and \( \rho \) is a weight parameter chosen by the user.

2.2 Measuring the diversity of solutions

A number of metrics exist for measuring the diversity of a set of solutions. A good metric needs to be model agnostic and ideally scaled such that diversity scores given by the metric are easy to interpret. Danna and Woodruff (2009) outlined three problem agnostic measures for the diversity of solutions - \( DBin \) defined in more detail below and used in our tests is scaled by the number of variables and solutions generated and considers just the binary variables, DAll which considers all variables types and DCV which is the scaled version of DAll.

The \( DBin \) metric (defined on the binary variables only) is the average scaled Hamming distance between all pairs of solutions in a set \( S \), i.e.,

\[ DBin(S) = \frac{2}{|S|(|S| - 1)} \sum_{j=1}^{|S|} \sum_{k=j+1}^{|S|} DBin(x^{(j)}, x^{(k)}) \]  (4)

where \( x^{(j)} \) is the \( j \)th solution generated by the oracle and \( DBin() \) computes the Hamming distance a pair of solutions, i.e.,

\[ DBin(x^{(j)}, x^{(k)}) = \frac{1}{|B|} \sum_{i \in B} |x^{(j)}_i - x^{(k)}_i| \]  (5)

where \( B \) is the set of binary variables and \( |B| \) is the number of binary variables. An advantage of the \( DBin \) metric is that it takes on values between 0 and 1 and does not depend on the size of the solution set or the number of variables.

3 Diversity-Emphasizing Node Selection rules

Within the branch-and-count algorithm (Achterberg et al. (2008)), we investigated several different variations of the best first search (BestFS) node selection that consider solution set diversity when selecting the next node to evaluate. We focused on BestFS because, as our results in §4 show, it generated the most diverse solution sets when compared with other well-known node selection rules (e.g., DFS) when diversity was not considered in the node selection task. What follows is a description of each of the custom node selection rules tested in this work.

3.1 Diverse-BFS (D-BFS(\( \alpha \)))

The Diverse-BFS (D-BFS \( \alpha \)) node selection rule considers both the lower bound of a node as well as the diversity of the node with respect to other solutions already in the near-optimal set. That is, this rule inputs a set of open nodes \( O \) and selects the next node \( i' \) according to:

\[ i' \in \arg \min_{i \in O} \{(1 - \alpha) L_i + \alpha D_i\}, \]

where \( \alpha \in [0, 1] \) is a parameter that trades off the bound of node \( i \) against its diversity score \( D_i \) and \( L_i \) is a scaled lower bound of node \( i \), i.e.,
\[
L_i = \frac{LB_i - \min_{j \in O} LB_j}{\max_{j \in O} LB_j - \min_{j \in O} LB_j}
\]

The lower bound is scaled to \([0, 1]\) because the diversity score \(D_i\) is in \([0, 1]\). The value of \(D_i\) represents the diversity of node \(i\) with respect to the current solution set \(\bar{S}\). For example, in the case of the DBin metric, we compute the average Hamming distance between node \(i\) and current solution set \(S\), only including the variables that are fixed at node \(i\) (set \(B_i\)) in the computation:

\[
D_i = \frac{1}{|S|/|B_i|} \sum_{j=1}^{|S|} \sum_{k \in B_i} |x_k^{(i)} - x_k^{(j)}|
\]

### 3.2 Diverse-BFS with tree depth (D-BFS(\(\alpha, \beta\)))

The D-BFS(\(\alpha, \beta\)) node selection rule considers the lower bound, the diversity and the depth of a node with respect to other solutions in the near-optimal set. This rule inputs a set of open nodes \(O\) and selects the next node \(i'\) according to:

\[
i' \in \arg \min_{i \in O} \{(1 - \alpha - \beta)L_i + \alpha D_i + \beta H_i\},
\]

where \(\alpha, (1 - \alpha - \beta), \beta \in [0, 1]\) are parameters that control the weight of the diversity score \(D_i\), the scaled lower bound of node \(i\), \(L_i\) as shown in 3.1 and the scaled depth of node \(i\), \(H_i\). Like \(L_i\), \(H_i\) is scaled as:

\[
H_i = \frac{\text{Depth}_i - \text{MinPlungeDepth}}{\text{MaxPlungeDepth} - \text{MinPlungeDepth}},
\]

where \(\text{MinPlungeDepth}\) and \(\text{MaxPlungeDepth}\) are set at the beginning of the computation and \(\text{Depth}_i\) is the depth of node \(i\) in the tree.

### 3.3 Diverse-BFS with Solution cutoff (D-BFS(\(\alpha, s\)))

The D-BFS(\(\alpha, s\)) rule considers only the lower bound of a node until it has generated a small set of solutions up to \(s\) before also considering the diversity of a node with respect to the solutions already in the near-optimal set. This rule selects the next node \(i'\) according to:

\[
i' \in \begin{cases} 
\arg \min_{i \in O} \{L_i\} & \text{number of solutions found so far } < s \\
\arg \min_{i \in O} \{(1 - \alpha)L_i + \alpha D_i\} & \text{otherwise}
\end{cases}
\]

where \(s\) is the solution cutoff parameter, i.e., the number of solutions that must be accumulated before diversity is considered in node selection.

### 3.4 Diverse-BFS with depth cutoff (D-BFS(\(\alpha, d\)))

The D-BFS(\(\alpha, d\)) rule considers only the lower bound of a node until the depth of the active node reaches a depth \(d\) before also considering the diversity of the node with respect to the solutions already in the near-optimal set. This rule selects the next node \(i'\) according to:

\[
i' \in \begin{cases} 
\arg \min_{i \in O} \{L_i\} & \text{depth of nodes in current iteration } < d \\
\arg \min_{i \in O} \{(1 - \alpha)L_i + \alpha D_i\} & \text{otherwise}
\end{cases}
\]
where \(d\) is the depth cutoff parameter, i.e. the depth that must be reached before diversity is considered in the node selection. If depth \(d\) is never reached, diversity is never triggered.

### 3.5 \textsc{DiversiT}ree - Diverse-BFS with solution cutoff and tree depth (D-BFS(\(\alpha, \beta, s\)) )

\textsc{DiversiT}ree selects the next node \(i'\) according to:

\[
   i \in \begin{cases} 
   \arg\min_{i \in O} \{L_i\} & \text{number of solutions found so far } < s \\
   \arg\min_{i \in O} \{(1 - \alpha - \beta)L_i + \alpha D_i + \beta H_i\} & \text{otherwise} 
\end{cases}
\]

where \(\alpha, \beta, s\) are parameters as defined in previous sections.

### 3.6 Other Diverse-BFS methods tested

We also tested several other methods for emphasizing solution set diversity. However, these additional methods were not as effective as the methods described in Sections 3.1 - 3.4 above. They are:

1. Using the minimum of diversity and depth, i.e., select next node \(i'\) according to:

   \[
   i' \in \arg\min_{i \in O} \{(1 - \alpha)L_i + \alpha(\min(D_i, H_i))\}
   \]

2. Using the maximum of diversity and depth, i.e., select next node \(i'\) according to:

   \[
   i' \in \arg\min_{i \in O} \{(1 - \alpha)L_i + \alpha(\max(D_i, H_i))\}
   \]

3. Using the product of diversity and depth, i.e., select next node \(i'\) according to:

   \[
   i' \in \arg\min_{i \in O} \{(1 - \alpha)L_i + \alpha D_i H_i\}
   \]

These node selections were also tested with a nonzero solution cutoff parameter, but were still not effective.

### 4 Results

To measure the effect of using diversity-emphasizing node selection rules on solution set diversity, we ran several sets of experiments on selected problems from MIPLIB (Koch et al. [2011], Bixby et al. [1998], Gleixner et al. [2021]) using several different node selection rules, including the customized ones described in §3. All code was implemented in C++ using SCIP Optimization suite 7.0 (Gamrath et al. [2020]) and run a server running Intel Xeon processors with sixteen cores and thirty two GB of memory. Apart from SCIP currently being one of the fastest non-commercial solvers for mixed integer programming (MIP) and mixed integer nonlinear programming, it provides a convenient way to use custom node selection rules. We evaluated our methods against the state-of-the-art \textsc{OneTree} method, implemented in GAMS-CPLEX.

Experiments were run to answer the following questions:

1. Among common node selection rules (e.g., BestFS and DFS), do some produce a more diverse set of near-optimal solutions than others? (§4.1)

2. What are the best parameters to use for the parameterized diversity-emphasizing node selection rules presented in Section 3? When using the best parameters, do the diversity-emphasizing node selection rules compute solution sets that are more diverse than those computed by competing approaches? (§4.2)

9
3. What parameters should be used when using diversity-emphasizing node selection rules on a new problem? (
§4.3)

4.1 Diversity of common node-selection rules

First, we examined the diversity produced by common node selection rules. For this first set of experiments, we selected problems from MIPLIB that had greater than 10,000 solutions within 1% of the optimal (see Table 3 in Appendix A).

Using a range of 50 to 2,000 requested near-optimal solutions, we used the common node selection rules listed in §2.1.1 to generate a set of solutions and computed the diversity. Figure 1 below shows the diversity ($DBin$) scores achieved by these common node selection rules for different solution set sizes ($p_1$). We labeled the different node selection rules with a B&C- prefix to indicate that we used branch-and-count to generate the solution sets. As shown in the figure, the best first search (B&C-BestFS) method finds sets with the largest diversity in 5 out of the 7 test problems. The depth first search (B&C-DFS) did not achieve the largest diversity in any of the instances. UCT (B&C-UCT) is best in 2 out of the 7 test problems and achieves a similar diversity value to hybrid estimate (B&C-HE) in all other cases.

The plots also indicate that diversity (aggregated over all problem instances) starts low and gets incrementally higher as the number of requested solutions increases. This is likely due to the fact that as more solutions are generated, more variables are fixed or modified; increasing the Hamming distance from the first solution found. However, this trend did not hold for all of the individual problem instances.
Figure 1: The final diversity ($DBin$) score achieved by common node selection rules (BestFS, DFS, UCT and HE) available on SCIP. We prefix each node selection rule name with “B&C” to indicate that we are using the “branch-and-count” method. For this test, we set $p_1 = p$. BestFS rule gave a higher diversity on more problems than other regular node selection methods.

Given that BestFS performed better than the other common rules, we used it for comparison in the remainder of our experiments.

4.2 Parameter optimization for diversity-emphasizing rules: training set

As described in §3, the diversity-emphasizing node selection rules have up to three parameters that must be tuned: when selecting between two solutions within the branch-and-bound procedure, $\alpha$ controls how diverse the selected solutions is, $\beta$ controls how deep within the search tree the selected solution is, and $S$ controls the number of solutions generated before utilizing the $\alpha$ and $\beta$ values in the solution selection process. For tuning we used the general three-parameter DiversiTREE method, which reduces to the other methods when some of its parameters are set to zero.

To tune the parameters we used a grid search to find the best-performing values of these parameters. Toward this end, we used several problem instances from MIPLIB (Gleixner et al. 2021, Koch et al. 2011, Bixby et al. 1998) and randomly split the set of problems into a training and testing set using a 75:25 split. We use the training set problems to
find the best parameters and then tested the performance of these parameters on the testing set problems. Using a grid-search on a 20x20x20 grid (each representing $\alpha$, $\beta$, and $s$ on the range 0.001, . . . 0.09, 0.1 , . . . 0.9, 1), we tested different parameters values across all the training set problems shown in Table 3 in Appendix A for different numbers of requested near optimal solutions $p_1$ (10, 50, 100, 200 and 1000) and values of $q$ (% near optimal) ranging from 1% - to 10%. As a result, we obtained the best performing values of $\alpha$, $\beta$, and $S$ for each instance and each value of $p_1$ and $q$.

We then ran a standard hierarchical clustering algorithm available in Python’s Scikit-Learn package \cite{Pedregosa2011} to cluster the problems into groups based on their best parameter settings, i.e., the settings that yielded the highest $DBin$ scores for that problem. When the number of requested solutions is small (i.e., 10), the clustering algorithm found the following four groups:

1. High $\alpha$, High $S$, Low $\beta$ (HHL): $\alpha \geq 0.9$, $S \geq 0.7$, $\beta \leq 0.2$.
2. High $\alpha$, Low $S$, Low $\beta$ (HLL): $\alpha \geq 0.9$, $S \leq 0.2$, $\beta \leq 0.2$.
3. Low $\alpha$, Low $S$, High $\beta$ (LLH): $\alpha < 0.2$, $S < 0.2$, $\beta \geq 0.8$.
4. Low $\alpha$, High $S$, High $\beta$ (LHH): $\alpha \leq 0.2$, $S \geq 0.7$, $\beta \geq 0.8$.

However, these four groups consolidate to three when the number of requested solutions reaches 50 and to two for 200 requested solutions or more.

Table 1 shows the grouping of the problem instances into the four groups. In the HHL group, diversity is emphasized heavily ($\alpha = 0.94$), but only after a large number of solutions have been accumulated ($S = 0.80$). In the HLL group, diversity is also heavily emphasized, but starting after a moderate number of solutions have been found ($S = 0.20$). The LLH group selects solutions deeper in the tree ($\beta = 0.99$) after a small number of solutions have been accumulated ($S = 0.05$). Finally, the LHH group emphasizes diversity lightly ($\alpha = 0.18$) and mostly selects solutions deeper in the tree ($\beta = 0.80$) after a large number of solutions have been accumulated ($S = 0.70$). As the table shows, the problems move to the group HLL as the number of requested solutions increases. Surprisingly, similar problems like stein27/stein45 and qnet1/qnet1_0 were not necessarily grouped into the same group.

The structure of these four groups indicates that our method is most effective when we generate a small number of seed solutions and then emphasize diversity deeper in the tree.

| Number of solutions requested | HHL | HLL | LLH | LHH |
|-----------------------------|-----|-----|-----|-----|
| 10                          | lseu, dcmulti, p0033, stein27, p0201, bell5, mod010, gt2, air03, fiber, gen, vpm1, vpm2, set1ch | mod008, p0548, 1152lav | stein45, rgn, misc06, misc03, qnet1, pp08a, pp08aCUTS, fixnet6 | khh05250, qnet1_o |
| 50                          | lseu, p0033, vpm1, bell5, qnet1, mod010, air03, set1ch, fiber, mod008, pp08aCUTS, dcmulti, pp08a | p0201, 1152lav, stein27 | rgn, misc06, gen gt2, p0548, misc03, qnet1_o, khh05250, fixnet6 | - |


DiversiTREE

Table 1 Continued

| 100 | 200 | 1000 |
|-----|-----|------|
| p0548, dcmulti, I152lav, fiber, lseu, p0033, pp08a, fixnet6, misc06, rgn | fiber, dcmulti, bell5, vpm1, gen | khhb05250 |
| bell5, stein27, mod010, air03, gen | lseu, mod008, p0033, pp08a, pp08aCUTS, khhb05250, gt2, misc03, mod008, qnet1, qnet1_o, vpm1, p0201 | lseu, mod008, p0033, pp08a, pp08aCUTS, gen, p0548, misc03, p0201, vpm1 |

Table 2: This are the α, β, and, S parameter setting used in both training and testing.

| Number of solutions requested | HHL | HLL | LLH | LHH |
|------------------------------|-----|-----|-----|-----|
| 10, 50, 100, 200, 1000       | α:0.94, β:0.06, s:0.80 | α:0.95, β:0.06, s:0.20 | α:0.01, β:0.99, s:0.05 | α:0.18, β:0.8, s:0.70 |

To assess the efficacy of DiversiTREE, we then selected a single parameter setting for each of the four groups, as shown in Table 2. Using the data from the grid search, we took the weighted average of the settings in a group as $G_1$ and the settings that occurred the most frequently in that group as $G_2$. We computed the diversity of the training set problems using $G_1$ and $G_2$ separately and selected either $G_1$ or $G_2$ as the best setting for the group, depending on which of the two produced the best diversity. We found that these parameter settings worked well for different values of $p_1$ and $q$.

Next, for each problem instance, we ran DiversiTREE with the parameters set according which of the four groups the problem was grouped into by the clustering algorithm (see Table 1). The parameter settings in Table 2 were used for all values of $p_1$ and $q$. Specifically, we used the following two-phase approach. In phase one, we generate a larger solution set with 10, 50, 100, 200, and 1000 solutions. In phase 2, we then use a subset selection method (similar to local search algorithm in Danna and Woodruff (2009)) to select $p = 10$ diverse solutions. (The value of 10 seems to be a reasonable number of solutions to present to a decision-maker). We ran this process for values of $q$ from 1% to 10% across all the problems in the training set. We compare our results to the B&C-BestFS rule and to results reported in Danna and Woodruff (2009).

In Figure 2, we plot the percent improvement of B&C-BestFS and the different groups of DiversiTREE over Onetree for values of $q$ from 1% to 10%, and $p = 10$ for different $p_1$ values. (In using B&C-BestFS, we found that the default parameters (i.e., MINPLUNGEDEPTH $= -1$, MAXPLUNGEDEPTH $= -1$, and MAXPLUNGEQUOT $= 0.25$) performed poorly. Thus, we manually tuned these settings to improve the diversity obtained by B&C-BestFS. These tuned parameter settings were used in the results shown in the remainder of the paper.)

As the figure shows, DiversiTREE achieves an improvement in diversity over Onetree of up to 160% and no worse
than 60%, no matter which of the four parameter settings were used. DIVERSITREE also significantly outperforms B&C-BestFS in terms of diversity for all values of $p_1$, since B&C-BestFS is only able to give an improvement of up to 104% and no worse than 42% over ONETREE. In terms of run time, when $p_1 < 100$ our method runs faster than B&C-BestFS and has comparable run time when $p_1 > 100$. Among the DIVERSITREE results, parameter groups HHL and HLL dominated the results.

**Figure 2:** The plots on the left show the average percent improvement on diversity (DBin) achieved by DIVERSITREE and B&C-BESTFS over ONETREE aggregated across all problems in the training set. The runtimes are shown in the plots on the right. The runtime for DIVERSITREE were similar for the four different parameter groups, hence this figure shows the average runtime over the four groups.
Figure 2: Cont... The plots on the left shows the average percent improvement on diversity ($DBin$) achieved by DiversiTREE and B&C-BESTFS over ONE_TREE aggregated across all problems in the training set. The runtimes are shown in the plots on the right. The runtime for DiversiTREE were similar for the four different parameter groups, hence this figure shows the average runtime over the four groups.

4.3 Performance of tuned parameter groups on test set of problems

The tuning results in the previous section raises the question of what parameters should be used on a new problem. To answer this question, we used the parameters from Table 2 (tuned using the training set of problems) on the set problems withheld in the 25% testing set (see Table 4 in Appendix B). We ran the same two-phase approach for diverse near optimal solution generation as we did in the training phase. We ran this process for $p_1 = 10, 50, 100, 200, 1000$ and $q$ from 1% to 10%. Again, the size of the final solution set was $p = 10$. The results achieved are shown in Figure 3, which displays results similar to those in Figure 2 for the training set. DiversiTREE generated solution sets with improvement over ONE_TREE of up to 139% and no worse than 32% no matter which of the four settings were used. In comparison, B&C-BestFS generated solution sets with improvement of up to 36% and no worse than 0%. In terms of runtime, DiversiTREE achieves similar run time as B&C-BestFS but the ONE_TREE method runs significantly faster than both methods. Unlike the training set data where parameter groups HHL and HLL dominated the DiversiTREE results, the parameter groups LHH and LLH dominated the diversity results generated in the test set.

In practice a business manager or process (say a software security strategy deployment tool) seeking a diverse set of near-optimal solutions would need to define $p$ (the number of near-optimal solutions), $q\%$ (how far these solutions should be from the optimal), and pick a setting in any of the groups we specified in §4.2 (without need for parameter tuning) and directly generate $p$ near-optimal solutions within $q\%$ of the optimal or use a two-phase approach to generate the solution set. The result of our tests suggest that the solution set generated would be no worse than B&C-BestFS and would be significantly better than current competing approaches.
Figure 3: Plots on the left show the percent improvement on diversity (DBin) achieved by DIVERSITREE and B&C-BESTFS on the problems in the test set using the tuned parameter settings shown in Table 2. The plots on the right show the run time. The runtime for DIVERSITREE were similar for the four different parameter groups, hence this figure shows the average runtime over the four groups.
Figure 3: Cont... Plots on the left show the percent improvement on diversity (DBin) achieved by DIVERSITREE and B&C-BESTFS on the problems in the test set using the tuned parameter settings shown in Table 2. The plots on the right show the run time. The runtime for DIVERSITREE were similar for the four different parameter groups, hence this figure shows the average runtime over the four groups.

5 Example: Railway Timetabling Problem

To provide a concrete example, we tested DIVERSITREE on a public transport scheduling problem within the MIPLIB2017 set (Gleixner et al., 2021). Details about the problem formulation and variables is available in Liebchen and Möhring (2003). Liebchen and Peeters (2002) discusses the general problem formulation. The problem models the cyclic railway timetabling problem where we have information about a railway network in a graph representing its infrastructure and traffic line. Each of the traffic line is operated every $T$ time units and the goal of the problem is to determine periodic departure times within the interval $[0, T]$ at every stop of every line. The problem has 397 variables; 77 of them are binary variables, 94 integer variables and 226 continuous variables. The final objective is to obtain an arrival/departure timetable with minimum passenger and vehicle waiting time. The problem is represented as a graph $G = (N, A, l, u)$ with nodes $N$ the sets of events, arcs $A$. The weight on the arcs represent the time when event $v_i \in N$ occurs and $[l, u]$ the allowable time interval for this event. The events are represented as triplets (train,node,arrival) or (train,node,departure) and the binary variable $b_{ij}$ is 1 if arc $(i, j)$ is selected. The model is mainly based on periodic constraints, which relate the arrival and departure time variables through the time window $[l, u]$ within which the event $i$ must occur.

For problems such as these, the solutions provided by the MIP model will be reviewed by a decision-maker prior to implementation. Thus, in this context it can be useful to provide the decision-maker with a set of near-optimal solutions to choose from. This will allow the decision-maker to consider not only the cost of the solution but also other factors such as variability in estimated demands, shortage of crew, behaviour of new equipment, or propagated delays, without having to re-solve the problem. For $q = 1\%$, the problem had 181 near optimal solutions and when $q$ increased to 3%, the number of near optimal solutions generated increased to 6,264.

We ran DIVERSITREE and the ONE.tree algorithm on this problem for $p_1 = 10, p = 10$ and $q = 3\%$. DIVERSITREE computed a solution set with a DBin value of 0.299, while ONE.tree’s set has a DBin value of 0.044. A further review of the generated solutions shows that of the 77 binary variables, ONE.tree gave a solution set in which 68 of the binary variables had the same value in ten all solutions. In addition, ONE.tree generated a set of ten solutions of which only six were unique. Among the six unique solutions, five of them had at most two variables with different values indicating very close similarity of the solutions. In constrast, DIVERSITREE’s gave a solution set in which only 23 of the variables were the same in all ten solutions, all ten solutions were unique and at least nine variables had different values among the solutions, indicating the diversity of the solutions. Thus, the DIVERSITREE solution set provides a much more diverse set of train schedules for a decision-maker to choose from.
6 Conclusion

The main goal of this article was to investigate whether emphasizing solution diversity in the node selection step of a branch-and-bound algorithm increases the diversity of the near-optimal solution set computed by the algorithm. Our results show that this is indeed the case, shown by our modified node selection rule that gives up to 140% better diversity than known methods for generating diverse solutions. Our method provides a fast tool for decision makers seeking a small number of near-optimal solutions to an optimization problem. We presented several methods for emphasizing diversity in node selection rules and optimal parameters that yield the most diverse set of solutions for the problems tested. We identified four groups of optimal parameters for different problems by clustering the parameter groups giving the best diversity (DBin) on the training set. Each group sets an optimal solution cutoff value and also sets how much diversity and depth to consider in determining the next solution added to the set of diverse solutions. When tested on new problems, our method (using the identified optimal parameters) ran in similar time as regular node selection rules and gave solution pools that were significantly more diverse.

6.1 Future work

A positive result of our study was that the four parameter setting groups identified during the parameter tuning on the training set performed well on a previously unseen set of problems in a test set. Even so, understanding the relationship between parameter settings and different classes of problems such that we can pick the best parameter for any problem with a minimal amount of tuning would be beneficial in practice. A second area of interest which is also an extension of this work is diversifying solution pools on select variables. In cases where a cluster of variables represent an attribute that is desirable in a machine learning algorithm (e.g., fairness, intelligence, etc.) it may be useful and informative to generate solutions pools that are diverse on only that attribute. A further extension of this is the ability to interpret and cluster the solution set based on desirable attributes such as fairness and intelligence. Finally, in this article, we only reported results using DBin as a diversity metric. It would be useful to understand how the new node selection method performs when using other metrics for computing diversity.

Acknowledgments

Medal gratefully acknowledges funding from the Army Research Office (grant W911NF-21-1-0079).

References

Tobias Achterberg, Stefan Heinz, and Thorsten Koch. Counting solutions of integer programs using unrestricted subtree detection. In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, Lecture Notes in Computer Science, pages 278–282, 2008. ISBN 9783540681540. doi: 10.1007/978-3-540-68155-7\_22.

Julien Baste, Michael R Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. Artificial Intelligence, 303:103644, 2022.

R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. Optima, 58:12–15, 1998.

Niclas Boehmer and Rolf Niedermeier. Broadening the research agenda for computational social choice: Multiple preference profiles and multiple solutions. In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, pages 1–5, 2021.

Qi Cai, Linwei He, Yimin Xie, Ruoqiang Feng, and Jianming Ma. Simple and effective strategies to generate diverse designs for truss structures. Structures, 32:268–278, 2021. ISSN 2352-0124. doi: 10.1016/j.istruc.2021.03.010.

Thomas R Cameron, Sebastian Charmot, and Jonad Pulaj. On the linear ordering problem and the rankability of data. arXiv preprint arXiv:2104.05816, 2021.
Sebastian Ceria, Cecile Cordier, Hugues Marchand, and Laurence A Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical programming*, 81(2):201–214, 1998.

Richard L Church and Carlos A Baez. Generating optimal and near-optimal solutions to facility location problems. *Environment and Planning B: Urban Analytics and City Science*, 47(6):1014–1030, 2020. ISSN 2399-8083. doi: 10.1177/2399808320930241.

Emilie Danna and David L Woodruff. How to select a small set of diverse solutions to mixed integer programming problems. *Operations Research Letters*, 37(4):255–260, 2009.

Emilie Danna, Mary Fenelon, Zonghao Gu, and Roland Wunderling. Generating multiple solutions for mixed integer programming problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 280–294. Springer, 2007.

Jonathan Eckstein. Parallel branch-and-bound methods for mixed integer programming. In *Applications on Advanced Architecture Computers*, pages 141–153. SIAM, 1996.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühlmeier, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL [http://www.optimization-online.org/DB_HTML/2020/03/7705.html](http://www.optimization-online.org/DB_HTML/2020/03/7705.html).

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarek, Thorsten Koch, Jeff Linderoth, Marco Litobecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. URL [https://doi.org/10.1007/s12532-020-00194-3](https://doi.org/10.1007/s12532-020-00194-3).

Fred Glover, Ching-Chung Kuo, and Krishna S. Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998. ISSN 0252-2667. doi: 10.1080/02522667.1998.10699366.

Fred Glover, Arne Løkketangen, and David L. Woodruff. Scatter search to generate diverse MIP solutions. In *Computing Tools for Modeling, Optimization and Simulation*, pages 299–317. Springer, 2000.

Peter Greist forfer, Arne Løkketangen, Stefan Voß, and David L. Woodruff. Experiments concerning sequential versus simultaneous maximization of objective function and distance. *Journal of Heuristics*, 14(6):613–625, 2008. ISSN 1381-1231. doi: 10.1007/s10732-007-9053-z.

Yunzhen He, Kun Cai, Zi-Long Zhao, and Yi Min Xie. Stochastic approaches to generating diverse and competitive structural designs in topology optimization. *Finite Elements in Analysis and Design*, 173:103399, 2020. ISSN 0168-874X. doi: 10.1016/j.finel.2020.103399.

V Roshan Joseph, Tirthankar Dasgupta, Rui Tuo, and CF Jeff Wu. Sequential exploration of complex surfaces using minimum energy designs. *Technometrics*, 57(1):64–74, 2015.

Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi: 10.1007/s12532-011-0025-9. URL [http://mpc.zib.de/index.php?IPC/article/view/56/28](http://mpc.zib.de/index.php?IPC/article/view/56/28).

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured MaxEnt RL. *Advances in Neural Information Processing Systems*, 33, 2020.

Ching-Chung Kuo, Fred Glover, and Krishna S Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):171–185, 1993.

Michael Lavine. Whim: function approximation where it matters. *Communications in Statistics-Simulation and Computation*, pages 1–31, 2019.

Christian Liebchen and Rolf H Möhring. Information on miplib’s timetab-instances. 2003.

Christian Liebchen and Leon Peeters. On cyclic timetabling and cycles in graphs. 2002.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.
F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pablo Rodríguez-Mier, Nathalie Poupin, Carlo de Blasio, Laurent Le Cam, and Fabien Jourdan. DEXOM: Diversity-based enumeration of optimal context-specific metabolic networks. *PLOS Computational Biology*, 17(2):e1008730, 2021. ISSN 1553-734X. doi: 10.1371/journal.pcbi.1008730.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022.

Patrick Schittekat and Kenneth Sörensen. OR practice—supporting 3PL decisions in the automotive industry by generating diverse solutions to a large-scale location-routing problem. *Operations Research*, 57(5):1058–1067, 2009.

Nicolas Schwind et al. Representative solutions for bi-objective optimisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1436–1443, 2020.

Lesia Semenova, Cynthia Rudin, and Ronald Parr. A study in rashomon curves and volumes: A new perspective on generalization and model simplicity in machine learning. *arXiv preprint arXiv:1908.01755*, 2019.

Thiago Serra. Enumerative branching with less repetition. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, pages 399–416, 2020. ISBN 9783030589417. doi: 10.1007/978-3-030-58942-4_26.

Thiago Serra and J. N. Hooker. Compact representation of near-optimal integer programming solutions. *Mathematical Programming*, 182(1-2):199–232, 2020. ISSN 0025-5610. doi: 10.1007/s10107-019-01390-3.

Rodothea Myrsini Tsoupidi, Roberto Castañeda Lozano, and Benoit Baudry. Principles and Practice of Constraint Programming, 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings. *Lecture Notes in Computer Science*, pages 791–808, 2020. ISSN 0302-9743. doi: 10.1007/978-3-030-58475-7_46.

Pascal Van Hentenryck, Carleton Coffrin, and Boris Gutkovich. Constraint-based local search for the automatic generation of architectural tests. In *International Conference on Principles and Practice of Constraint Programming*, pages 787–801. Springer, 2009.

Bo Wang, Yan Zhou, Yiming Zhou, Shengli Xu, and Bin Niu. Diverse competitive design for topology optimization. *Structural and Multidisciplinary Optimization*, 57(2):891–902, 2018.

Tom Zahavy, Brendan O’Donoghue, Andre Barreto, Volodymyr Mnih, Sebastian Flennerhag, and Satinder Singh. Discovering diverse nearly optimal policies with successor features. *arXiv preprint arXiv:2106.00669*, 2021.

Elena Zaslavsky and Mona Singh. A combinatorial optimization approach for diverse motif finding applications. *Algorithms for Molecular Biology*, 1(1):1–13, 2006. doi: 10.1186/1748-7188-1-13.

Yiming Zhou, Raphael T. Haftka, and Gengdong Cheng. Balancing diversity and performance in global optimization. *Structural and Multidisciplinary Optimization*, 54(4):1093–1105, 2016. ISSN 1615-147X. doi: 10.1007/s00158-016-1434-1.
Appendix A  Problems in the training set

Table 3 captures the problems we solved in the training set. They are randomly selected problems from MIPLIB (Gleixner et al. (2021), Koch et al. (2011), Bixby et al. (1998)). We did remove instances that did not complete computation of the objective value within 30 minutes. In total, 27 problem instances are used in this training set. We capture the characteristics of the problem below.

Table 3: The problem instances used for training in Section 4.1 and 4.2 and their characteristics are shown in this table.

| Problem Instance | Total variables | Binary variables | General integer variables | Continuous variables | Solutions within 1% of the optimum |
|------------------|-----------------|------------------|--------------------------|-----------------------|-----------------------------------|
| air03            | 10757           | 10757            | 0                        | 0                     | 938                               |
| bell5            | 104             | 30               | 28                       | 46                    | >10,000                           |
| dcmulti          | 548             | 75               | 0                        | 473                   | >10,000                           |
| fiber            | 1298            | 1254             | 0                        | 44                    | >279                             |
| fixnet6          | 878             | 378              | 0                        | 500                   | >10,000                           |
| gen              | 870             | 144              | 6                        | 720                   | >10,000                           |
| gesa3            | 1152            | 216              | 168                      | 768                   | >10,000                           |
| gt2              | 188             | 24               | 164                      | 0                     | >10,000                           |
| khb05250         | 1350            | 24               | 0                        | 1326                  | 28                               |
| l152lav          | 1989            | 1989             | 0                        | 0                     | >10,000                           |
| misc03           | 160             | 159              | 0                        | 1                     | 24                               |
| misc06           | 1808            | 112              | 0                        | 1696                  | >10,000                           |
| mod008           | 319             | 319              | 0                        | 0                     | 68                               |
| mod010           | 2655            | 2655             | 0                        | 0                     | >10,000                           |
| p0033            | 33              | 33               | 0                        | 0                     | 15                               |
| p0201            | 201             | 201              | 0                        | 0                     | 44                               |
| p0548            | 548             | 548              | 0                        | 0                     | >10,000                           |
| pp08a            | 240             | 64               | 0                        | 176                   | 64                               |
| pp08aCUTS        | 240             | 64               | 0                        | 176                   | 64                               |
| qnet1            | 1541            | 1288             | 129                      | 124                   | >10,000                           |
| qnet10           | 1541            | 1288             | 129                      | 124                   | >10,000                           |
| rgn              | 180             | 100              | 0                        | 90                    | >720                             |
| set1ch           | 712             | 240              | 0                        | 472                   | >10,000                           |
| stein27          | 27              | 0                | 0                        | 0                     | 2106                             |
| stein45          | 45              | 0                | 0                        | 0                     | 70                               |
| vpm1             | 378             | 168              | 0                        | 210                   | >10,000                           |
Appendix B  Problems in the testing set

Table 4 captures the problems we solved in the testing set. They are randomly selected problems from MIPLIB (Gleixner et al. (2021)). We did remove instances that did not complete computation of the objective value within 30 minutes. In total, 9 problem instances are used in this training set. We capture the characteristics of the problem below.

Table 4: The problem instances used for training in Section 4.1 and 4.2 and their characteristics are shown in this table.

| Problem Instance | Total variables | Binary variables | General integer variables | Continuous variables | Solutions within 1% of the optimum |
|------------------|-----------------|------------------|---------------------------|----------------------|------------------------------------|
| 23588            | 368             | 231              | 0                         | 137                  | 82                                 |
| bppc8-02         | 232             | 229              | 1                         | 2                    | >10,000                            |
| exp-1-500-5-5    | 990             | 250              | 0                         | 740                  | >1,338                             |
| mtest4ma         | 1950            | 975              | 0                         | 975                  | >10,000                            |
| neos-1425699     | 105             | 5                | 80                        | 20                   | >10,000                            |
| neos17           | 535             | 300              | 0                         | 235                  | >10,000                            |
| nexp-50-20-1-1   | 490             | 245              | 0                         | 245                  | >10,000                            |
| sp150x300d       | 600             | 300              | 0                         | 300                  | >9,455                             |