

Table of Contents for Appendix

A	Proof of Permutation Equivalence	23
A.1	Formulation for Transformer	23
A.2	Proof of Proposition 1	23
B	Implementation Details of Separator Configuration in SCIP	24
B.1	Frequency and Priority Setup in SCIP	24
B.2	Input Features of the Triplet Graph	25
B.3	Separator Statistics	25
B.4	Stopping Conditions for a Separation Loop	27
B.5	Separators Built In SCIP	27
C	Network Details	29
C.1	The Encoder Network	29
D	Implementation Details of our algorithm	29
D.1	Algorithm Pseudocode of DynSep	29
D.2	Hyperparameters	29
D.3	Hardware Specification	31
D.4	Baselines	31
E	Additional Results	31
E.1	Motivation Results on effects of different configurations	31
E.2	Evaluation Results on Other Metrics	32
E.3	Evaluation on Additional MIPLIB Datasets	33
E.4	Extended DynSep to Broader Solver Hyperparameters	34
E.5	Overhead Evaluation	35
E.5.1	Latency of Policy Inference	35
E.5.2	Memory Overhead	35
E.6	Ablation Study	36
E.6.1	Module Ablation Analysis on Other Six Benchmarks	36
E.6.2	Encoder Architecture Ablation	36
E.6.3	Hyperparameter Sensitivity Analysis	36
E.7	Generalization Study	37
E.7.1	Cross-Domain Generalization Test	37
E.7.2	General-to-Specific Generalization Test	37
E.8	Visualization of Separator Configurations on Nine Benchmarks	38

A Proof of Permutation Equivalence

A.1 Formulation for Transformer

The standard Transformer architecture comprises two main components: the multi-head self-attention layers (MHA) and the position-wise feed-forward network (FFN). In the following part, we will briefly introduce these blocks. We represent an input sequence as $\mathbf{X} = \langle \mathbf{x}_1, \dots, \mathbf{x}_N \rangle \in \mathbb{R}^{N \times d}$, where \mathbf{x}_i is the hidden embedding for the token i , and d is the dimension of the embeddings. The MHA module projects \mathbf{X} to a triplet $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, as follows.

$$\mathbf{Q}_X = \mathbf{X}\mathbf{W}_Q, \mathbf{K}_X = \mathbf{X}\mathbf{W}_K, \mathbf{V}_X = \mathbf{X}\mathbf{W}_V,$$

$$\text{Attention}(X) = \text{softmax}\left(\frac{\mathbf{Q}_X \mathbf{K}_X^\top}{\sqrt{d_K}}\right) \mathbf{V}_X,$$

where $\mathbf{W}_Q \in \mathbb{R}^{d \times d_K}$, $\mathbf{W}_K \in \mathbb{R}^{d \times d_K}$, $\mathbf{W}_V \in \mathbb{R}^{d \times d_V}$ are learnable weights, with $d_K = d_V = \frac{d}{H}$. Overall, H such projections are performed, resulting in $(\mathbf{Q}_X^{(h)}, \mathbf{K}_X^{(h)}, \mathbf{V}_X^{(h)})$ for $1 \leq h \leq H$. The self-attention operation is then applied to each triplet:

$$\text{head}_h = \text{softmax}\left(\frac{\mathbf{Q}_X^{(h)} \mathbf{K}_X^{(h)\top}}{\sqrt{d_K}}\right) \mathbf{V}_X^{(h)}, \quad (8)$$

$$\text{MHA}(\mathbf{H}) = \text{concat}(\text{head}_1, \dots, \text{head}_H) \mathbf{W}_O, \quad (9)$$

where $\mathbf{W}_O \in \mathbb{R}^{d \times d}$ is a learnable weight matrix. The output of the MHA module is then passed through a feed-forward network layer, followed by a residual connection and layer normalization (LN). Finally, the output of the l -th layer \mathbf{H}^l is computed as follows:

$$\hat{\mathbf{H}}^l = \text{LN}(\mathbf{H}^{l-1} + \text{MHA}(\mathbf{H}^{l-1})), \quad (10)$$

$$\mathbf{H}^l = \text{LN}(\hat{\mathbf{H}}^l + \text{FFN}(\hat{\mathbf{H}}^l)). \quad (11)$$

To remain consistent with the notation of the common Transformer study, a small subset of definitions in this appendix overlaps with those in the main text. We claim that these symbol definitions are valid only within this chapter.

A.2 Proof of Proposition 1

We now present the full proof of Proposition 1. Recalling that the blocked positional encoding is defined by

$$P = [P_0^\top \ P_1^\top \ \dots \ P_T^\top]^\top \in \mathbb{R}^{(T+1)(K+1) \times d}, \quad (12)$$

where $P_i = [\text{pos}(i) \ \text{pos}(i) \ \dots \ \text{pos}(i)]^\top \in \mathbb{R}^{(K+1) \times d}$, $\text{pos}(i) \in \mathbb{R}^d$, so that each of the $K+1$ tokens in block i shares the same vector $\text{pos}(i) \in \mathbb{R}^d$.

Proposition. *The decoder-only Transformer \mathcal{T} , equipped with the blocked positional encoding P , is permutation equivariant inside each token block. Formally, for any input X and any block-wise permutation matrix Π , it holds that $\mathcal{T}(\Pi X + P) = \Pi \mathcal{T}(X + P)$.*

Proof. Because feed-forward, layer-normalization, and residual addition each of them either applies elementwise or row-wise operations, or multiplies on the right by a learnable weight matrix, they automatically commute with any row-permutation of their input. Hence, to prove that the full model is equivariant under block-wise token permutations, it suffices to show that each self-attention module satisfies $\text{Attention}(\Pi X + P) = \Pi \cdot \text{Attention}(X + P)$.

We define the block-wise permutation matrix Π as a permutation matrix that operates independent permutations within each block, which is formulated as follows:

$$\Pi = \begin{bmatrix} \Pi_0 & & & \\ & \Pi_1 & & \\ & & \ddots & \\ & & & \Pi_{t-1} \end{bmatrix},$$

where $\Pi_i \in \{0, 1\}^{(K+1) \times (K+1)}$ is an arbitrary permutation matrix, with the condition that each row and each column contains exactly one entry of 1 and the rest are 0. Denote the input by:

$$X = [z_0^\top \quad z_1^\top \quad \cdots \quad z_{t-1}^\top]^\top \in \mathbb{R}^{(K+1)t \times d}, \text{ where } z_i = [h_{i,0} \quad h_{i,1} \quad \cdots \quad h_{i,K}]^\top \in \mathbb{R}^{(K+1) \times d}.$$

Apply Π yields $\tilde{X} = \Pi X = [\Pi_0 z_0^\top \quad \Pi_1 z_1^\top \quad \cdots \quad \Pi_{t-1} z_{t-1}^\top]^\top$. Because each block P_i in our positional encoding consists of identical rows, permitting these rows does no change, i.e., $\Pi_i P_i = P_i$ for any i . Hence, we have that

$$\Pi X + P = \Pi X + \Pi P = \Pi(X + P). \quad (13)$$

Next, after performing the blocked positional encoding, we have the attention of \tilde{X} as follows:

$$\text{Attention}(\tilde{X} + P) = \text{softmax} \left(\frac{(\Pi X + P) \mathbf{W}_Q \mathbf{W}_K^\top (\Pi X + P)^\top}{\sqrt{d_K}} \right) (\Pi X + P) \mathbf{W}_V \quad (14)$$

Since softmax is applied row-wise, it can be viewed as left-multiplying the input by a matrix. Thus, by the associativity of matrix multiplication, we can freely regroup the products of the matrices in the above formula (14). First, we deduce that

$$\begin{aligned} (\Pi X + P)^\top (\Pi X + P) &= \sum_{i=0}^t (\Pi_i z_i + P_i)^\top (\Pi_i z_i + P_i) \\ &= \sum_{i=0}^t z_i^\top \Pi_i^\top \Pi_i z_i + X_i^\top \Pi_i^\top P_i + P_i^\top \Pi_i X_i + P_i^\top P_i \\ &= \sum_{i=0}^t z_i^\top z_i + X_i^\top P_i + P_i^\top X_i + P_i^\top P_i \end{aligned} \quad (15)$$

$$\begin{aligned} &= \sum_{i=0}^t (z_i + P_i)^\top (z_i + P_i) \\ &= (X + P)^\top (X + P), \end{aligned} \quad (16)$$

where the deduction of equation (15) utilizes the property that $\Pi^\top \Pi = \mathbf{I}$ and $\Pi_i^\top P_i = P_i \Pi_i^\top = P_i$.

Therefore, substituting equations (16) and (13) into (14), we have

$$\text{Attention}(\tilde{X} + P) = \Pi \cdot \text{softmax} \left(\frac{(X + P) \mathbf{W}_Q \mathbf{W}_K^\top (X + P)^\top}{\sqrt{d_K}} \right) (X + P) \mathbf{W}_V \quad (17)$$

$$= \Pi \cdot \text{softmax} \left(\frac{\mathbf{Q}_X \mathbf{K}_X^\top}{\sqrt{d_K}} \right) \mathbf{V}_X \quad (18)$$

$$= \Pi \cdot \text{Attention}(X + P). \quad (19)$$

Finally, each subsequent layer (feed-forward, layer norm, residual connections) also commutes with block-wise permutation, so the entire model satisfies the block-wise permutation equivariance:

$$\mathcal{T}(\Pi X + P) = \Pi \mathcal{T}(X + P),$$

as required. \square

B Implementation Details of Separator Configuration in SCIP

B.1 Frequency and Priority Setup in SCIP

In the SCIP solver, we adjust the activation status configuration by two hyperparameters: the frequency f_i and the priority q_i . We provide the detailed description of these two configuration parameters as follows.

SEPA_FREQ f_i : The frequency parameter determines at which nodes in the branch-and-bound tree a separator is invoked. Specifically, setting the $f_i = -1$ disables the separator entirely while $f_i = 0$ activates the separator in any separation round of any tree node. Any positive $f_i > 0$ activates the

separator at every node whose depth is a multiple of f_i . We set $f_i = 10$ for all active separators in our paper.

SEPA_PRIORITY p_i : The priority parameter dictates the order in which separators are executed during a separation round at a node. In every separation round, all separators with $p_i \geq 0$ are executed first in descending order of p_i , then constraint handlers are applied, and finally separators with $p_i < 0$ run in descending order of p_i . By convention, separators implementing fast, high-impact cuts have large non-negative priorities so that their cuts are added early, thus strengthening the LP relaxation sooner. In contrast, more expensive or specialized separators are given negative priorities. Hence, they run later or only if no earlier cuts were found, thereby avoiding unnecessary overhead at the start of each node’s separation phase. This division into early ($p_i \geq 0$) versus late ($p_i < 0$) activation phases directly influences the quality of intermediate bounds: running aggressive separators early can dramatically tighten the relaxation and reduce the number of branch-and-bound nodes, while deferring or disabling them can save CPU time when their benefit is marginal at that stage.

Activation Status To encapsulate the activation configuration, we define an activation status variable η_i for each separator. In detail, $\eta_i = 0$ means that we set the frequency of separator $f_i = 0$ and thus let it never run in all B&B tree nodes. $\eta_i = +1$ means that we set $f_i > 0$, but the priority of the separator $p_i \geq 0$, executing it before the constraint handler [44]. $\eta_i = -1$ means that we set $f_i > 0$ and $p_i < 0$, which means the separator is activated but executed after the constraint handler. As shown by the orange bar in Fig. 2(b) of the main text, perturbation in priority configs exhibits a minor effect on performance improvement compared to activation status (yellow bars) changes. This phenomenon arises because activation status dictates the order of separators across rounds, whereas priority affects only their relative order within a round; since the LP is not re-solved until the end of a round, reordering separators within the same round has little effect on overall performance.

B.2 Input Features of the Triplet Graph

Node Features Each node type—variables, constraints, and separators—is characterized by a set of features that encapsulate their properties and roles within the optimization process. The input features for variable nodes V , constraint nodes C , and separator nodes are list in Table 4. Furthermore, we incorporate two graph-level input features—the dual degeneracy rate and the variable-to-constraint ratio. Concretely, the dual degeneracy rate is the fraction of nonbasic variables having zero reduced cost, and the variable-to-constraint ratio is the number of unfixed variables relative to the LP basis size. In practice, we first apply an additive (sum) pooling over the GCN encoder’s node representations to obtain a single aggregated node feature vector. We then append (concatenate) the two scalar metrics to this pooled embedding. The combined vector is passed through a multilayer perceptron, yielding a unified graph embedding that fuses the local structural information.

Edge Features Like the bipartite graph modeling for common MILP problem, we construct edges between variable and constraint nodes such that a variable node V_i is connected to a constraint node C_j if the variable appears in the constraint with the weight corresponds to the coefficient $A_{ij} \neq 0$, and we set the value of the edge as A_{ij} .

B.3 Separator Statistics

In SCIP’s solver-statistics display, each separator reports several key metrics that provide insights into its performance and impact during the solving process. We use these metrics as input features of separator nodes and immediate reward signals at each time step. These metrics include:

Cut Application Rate. This rate measures the effectiveness of a separator’s generated cuts by calculating the ratio of cuts applied to the LP relaxation to the total number of cuts found. The application rate is computed as:

$$\text{Cut Application Rate} = \frac{\text{Number of Cuts Applied}}{\text{Number of Cuts Found}} \quad (20)$$

A higher application rate suggests that the separator frequently generates cuts deemed valuable and effective by SCIP’s internal filtering mechanisms, leading to their inclusion in the LP relaxation. Conversely, a lower rate may indicate that many of the separator’s cuts are redundant or less impactful.

Domain Reductions (DomReds) Separators can also perform domain reductions by tightening variable bounds through their logic (for example, by deducing $x_i \leq u$ or $x_j \geq l$ from cut coefficients).

Table 4: Description of input features for variable, constraint, separator nodes, and the entire graph.

Type	Feature	Setting
Vars	type	Variable’s type: binary, integer, implicit integer, continuous (one-hot).
	has_lb	If the variable has an infinite lower bound.
	lb	The variable’s lower bound, set 0 if it is infinite.
	has_ub	If the variable has an infinite upper bound.
	ub	The variable’s upper bound, set 0 if it is infinite.
	basestat	Simplex basis status: lower, basic, upper, zero (one-hot)
	coef_norm	Objective coefficient, normalized by objective norm
	reduced_norm	Variable’s reduced cost divided by the objective norm, indicating how much the objective would worsen per unit increase at zero slack
	age	The number of consecutive LP iterations in which the variable stayed at zero in the basis.
	solval	The primal LP solution value of the variable.
	solfrac	The fractional part of ‘solval’.
Cons	sol_is_at_lb	If ‘solval’ equals the lower bound within numerical tolerance
	sol_is_at_ub	If solval equals the upper bound within numerical tolerance
	round_num	Index of the current separation round
	origin_type	Which mechanism generated this row: unspecified, constraint handler, constraint, separator, reoptimization (one-hot).
	origin_sepa	The separator name that produced this row.
	basestat	The row’s basis status in the LP solution: basic, lower, upper (one-hot).
	bias	Unshifted side normalized by row norm.
	dualsol	Dual LP solution of a row, normalized by row and objective norm.
	is_at_lhs	If the row value equals the left-hand side.
	is_at_rhs	If the row value equals the right-hand side.
	norm_nnzr	Number of nonzero coefficients in the row, normalized by the total number of LP variables.
Seps	age	The count of successive LP iterations for which the row has stayed nonbasic at zero.
	int_support	Integral support score of a row.
	is_integral	Activity of the row is always integral in a feasible solution.
	is_removable	Row is removable from the LP.
	is_in_lp	Row is member of current LP.
	round_num	Index of the current separation round
	round_num	Index of the current separation round
Seps	type	Type of the separator (one-hot).
	time	Execution time consumed by the separator.
	calls	Number of times that the separator has been invoked.
	cuts	Number of cuts generated by this separator.
	cutoffs	Number of infeasibility detections (cutoffs) found by the separator.
	domreds	Number of domain reductions found by the separator.
Graph	applied	Number of cuts from the separator that were applied to the LP relaxation.
	dual_deg_rate	The proportion of nonbasic variables with reduced cost zero.
Graph	var_con_ratio	The ratio of unfixed variables to the size of the LP basis.

Each time a separator callback successfully reduces a variable’s domain, the DomReds counter is incremented. This statistic captures the total number of such bound-tightening operations executed by that separator during the solve. A higher DomReds count signifies that the separator contributes significantly to shrinking feasible regions, which can indirectly improve future LP relaxations and cut generation efficiency.

Cutoffs. Within each separation round, when a separator generates one or more cuts that render the LP infeasible or whose bound surpasses the current best primal solution, SCIP immediately prunes that node (i.e., cuts it off) without further processing. The Cutoffs statistic for a separator is simply the total count of these pruned nodes attributable to its cuts. In practice, a high Cutoffs value indicates

that the separator is highly effective at early fathoming of unpromising subproblems, potentially reducing the size of the branch-and-bound tree.

B.4 Stopping Conditions for a Separation Loop

In SCIP, the separation process at a node is conducted in iterative rounds, where each round involves generating cutting planes to refine the LP relaxation. The separation loop at a node terminates when any of the following conditions are met:

Maximum Number of Rounds Reached: A user-defined limit on the number of separation rounds per node is enforced to prevent excessive computation. For experimental stability, we use only the maxround m_t decision from the first separation round at each node to set the maximum number of subsequent separation rounds at the current node.

Stalling Criterion Triggered: If consecutive separation rounds fail to yield improvements in the objective bound or integrality, the process is considered to have stalled, prompting termination.

Relative Bound Distance Exceeded: Separation is halted if the relative distance between the current node's dual bound and the global primal bound surpasses a predefined threshold, indicating diminishing returns from further separation.

No Further Separation Requested: If all separators and constraint handlers indicate that no additional separation is necessary (i.e., none return a status requesting another round), the loop concludes.

These stopping criteria ensure a balance between the thoroughness of the separation process and computational efficiency, preventing unnecessary iterations that offer minimal benefit to the overall solution process.

B.5 Separators Built In SCIP

We consider 22 separators in our configuration task. We provide the detailed description of these separators as follow.

closecuts. Close cuts are a type of cutting plane that focuses on generating cuts that are "close" to the current fractional solution. These cuts are designed to tighten the feasible region by targeting solutions that are near the boundary of the current relaxation. The idea is to improve the quality of the LP relaxation by adding cuts that are particularly relevant to the current solution.

disjunctive. Disjunctive cuts are a class of cutting planes used in mixed-integer programming, particularly based on the concept of disjunctions. These cuts are derived from a disjunctive argument that partitions the solution space into different disjunctive sets. By analyzing the infeasible or fractional solutions that arise from linear relaxations, disjunctive cuts can tighten the formulation by excluding these solutions and enforcing integrality conditions more strongly.

minor. Derived from graph minor theory, these cuts identify isomorphic substructures in the constraint matrix corresponding to known hard combinatorial subproblems. By recognizing these patterns, the separator generates cuts that exploit the inherent complexity of the substructures.

mixing. Generates cuts by combining multiple weak constraints through coefficient mixing, creating stronger aggregated inequalities. The method systematically blends constraints sharing common variable structures while preserving problem feasibility.

rlt. Reformulation-Linearization Technique (RLT) converts polynomial constraints into linear inequalities through variable substitution and constraint multiplication. RLT preserves problem structure while creating convex envelopes for nonlinear terms, enabling strong linear relaxations.

interminor. This separator extends minor cuts by focusing on medium-scale substructures that appear as intermediate components in larger mixed-integer programming (MIP) models. Balances local pattern matching with global problem structure analysis.

convexproj. Convex projection cuts are generated by projecting an infeasible point onto a convex relaxation of the problem and then creating gradient-based cuts at the projection point. These cuts are designed to improve the separation of fractional solutions in convex nonlinear programs. The method

aims to enhance the solver's ability to make progress by refining the feasible region through gradient information at the projected point.

gauge. Geometric cuts based on analyzing the gauge function of the feasible region's convex hull. This separator generates deep cuts orthogonal to the objective gradient by exploiting polyhedral geometry.

impliedbounds. Implied bound cuts are cutting planes that leverage implications between binary and continuous variables to restrict the feasible region of an MILP problem. They enforce tighter constraints by exploiting logical relationships, such as when a binary variable limits a continuous variable's upper or lower bound.

intobj. Integer objective cuts are cutting planes used in MIP when the objective function is integer-valued. These cuts aim to eliminate fractional solutions from the LP relaxation that are not feasible in the integer solution space. They help tighten the LP relaxation by leveraging the fact that certain fractional values of the objective function cannot lead to valid integer solutions.

gomory. Gomory cutting planes are derived from the fractional solutions of the LP relaxation of an MIP problem. Once the LP relaxation is solved, any variable with fractional values can be targeted, and a Gomory cut is generated to eliminate these fractional solutions, moving the solution closer to integrality. These cuts can be generated iteratively during the branch-and-bound process to progressively tighten the LP relaxation.

cgmip. Chvatal-Gomory cuts are generated by forming non-negative integer combinations of the original linear constraints and then rounding the resulting coefficients to produce a valid inequality. These cuts are designed to tighten the LP relaxation by eliminating fractional solutions. The process involves solving a sub-MIP to identify the best combination of constraints, which ensures that the generated cuts are as effective as possible.

strongcg. Strong Chvátal-Gomory (CG) cutting planes are an extension of the classical CG cuts, which are derived from valid inequalities of the linear relaxation of an integer programming problem. These cuts are used to iteratively tighten the LP relaxation by adding inequalities that exclude fractional solutions. The strong variant refers to CG cuts that are particularly effective in reducing the feasible region, leading to a faster convergence to the integer solution.

aggregation. This separator generates cuts by aggregating multiple constraints into single strengthened inequalities. It specializes in flow cover inequalities for network problems, combining arc selection variables with flow conservation constraints.

clique. Clique cutting planes are a type of valid inequality derived from the set-packing formulation in integer programming, particularly useful in problems involving binary variables. The inequalities are based on identifying cliques in a conflict graph representation of the problem. A clique is a subset of mutually adjacent vertices in a graph, representing a set of constraints that cannot be simultaneously satisfied. The corresponding clique cutting planes exclude infeasible solutions by enforcing that only one element from each clique can be selected.

zerohalf. Zero-half cuts are a specific type of Chvátal-Gomory (CG) cuts in integer programming. These cuts are derived using coefficients in $\{0, \frac{1}{2}\}$ instead of integer coefficients. They are used to tighten the relaxation of integer programming problems, bringing it closer to the convex hull of feasible integer solutions.

mcf. Flow path cuts are valid inequalities used to strengthen the linear relaxation of MIP problems, specifically for problems involving fixed charge networks. They help in modeling flow through a sequence of nodes where fixed charges are incurred if any flow occurs along a path. Flow path inequalities operate on constraints related to fixed charge paths, where binary and continuous variables govern the flow through a network. These cuts are particularly useful in fixed charge network design and lot-sizing problems. However, the computational consideration is that the structure exploited by these cuts is very specific, meaning they are only applicable to certain problem types.

eccuts. This separator specializes in cuts for edge-concave functions in Mixed-Integer Nonlinear Programming (MINLP), generated by constructing supporting hyperplanes at concave function edges. It exploits piecewise-linear approximations of nonlinear constraints.

oddcycle. Odd cycle cuts are designed to eliminate infeasible fractional assignments in problems where binary variables represent nodes or edges in a graph. They are particularly effective when

dealing with cycles in a graph that contain an odd number of nodes. For example, in a graph-based problem, assigning fractional values to all variables in an odd cycle is infeasible when the solution must be binary (0 or 1). Odd cycle cuts ensure that such fractional solutions are excluded from the feasible region.

flowcover. Flow cover cuts are a type of cutting plane derived from valid inequalities used to tighten the linear relaxations of MIP problems, particularly for binary single-node flow sets. They are useful in network design and fixed charge problems, where variables can represent flows subject to upper bounds and binary decisions.

cmir. MIR cuts are a class of cutting planes derived from mixed-integer sets, particularly when dealing with constraints that include both continuous and integer variables. The main idea behind MIR cuts is to generate valid inequalities by rounding coefficients of mixed-integer constraints to tighten the LP relaxation. They are generated using a disjunctive argument, which creates inequalities that separate fractional solutions from the feasible region of the MIP.

rapidlearning. Rapid learning is a heuristic technique that temporarily relaxes certain constraints or simplifies the problem to solve a more manageable version. By solving this easier problem, the solver gains insights into the structure of the original problem. The rapid learning separator then uses this information to generate useful cuts or constraints that can immediately improve the quality of the LP relaxation in the original problem.

C Network Details

C.1 The Encoder Network

We provide a detailed description of the neural architecture employed in our encoder network. Our design builds upon the framework introduced in L2Sep [6], incorporating several modifications to enhance performance. The encoder first embeds maps the input features of constraint (C), variable (V), and separator (S) nodes into hidden representations. Subsequently, it performs message passing following the directions of $V \rightarrow C \rightarrow V$, $S \rightarrow V \rightarrow S$, and $S \rightarrow C \rightarrow S$, effectively capturing the interactions among different node types. Then, the S nodes pass through an attention module to emphasize the task of the separator configuration. In contrary to the approach in [45], which outputs a score for each cut node, our encoder applies a global additive pooling on each of the C , V , and S hidden embeddings, yielding three aggregated embedding vectors. These vectors are concatenated with two graph-level features, as detailed in Section B.2, forming a comprehensive representation. Finally, this combined vector is passed through a multilayer perceptron (MLP) to produce a unified graph embedding that encapsulates both local and global information pertinent to the problem structure.

D Implementation Details of our algorithm

D.1 Algorithm Pseudocode of DynSep

We employ the Proximal Policy Optimization (PPO) algorithm to train our model. PPO alternates between collecting data through interactions with the environment and optimizing a surrogate objective function to update the policy. To ensure stable training, PPO utilizes a clipped surrogate objective that constrains the policy updates, preventing drastic changes that could destabilize learning. Specifically, the policy network is updated to maximize the expected advantage while maintaining the probability ratio between the new and old policies within a predefined threshold. Concurrently, the value network is trained to minimize the mean squared error between the predicted value estimates and the actual returns. The training procedure of our method DynSep, including the formulation of PPO objective functions, is detailed in the pseudocode of Algorithm 1.

D.2 Hyperparameters

We train DynSep using the ADAM optimizer [46] within the PyTorch framework [47]. Consistent with prior studies [33, 42], we split each dataset into the train and test sets with 80% and 20% instances. We train our model on the train set for 100 epochs, and select the best model on the train set to evaluate on the test set. A complete list of hyperparameters for the SCIP solver, the PPO algorithm, and the decoder-only Transformer appears in Table 5.

Algorithm 1 Dynamic Separator Configuration via PPO (DynSep)

```

1: Denote parameters of the actor's transformer  $\mathcal{T}_\pi$  and policy  $\pi$  as  $\theta$ .
   Denote parameters of the critic's transformer  $\mathcal{T}_V$  and value function  $V$  as  $\psi$ .
   Denote parameters of the encoder  $\Phi$  as  $\beta$ .
2: Initialize MILP instances set  $\mathcal{X}$ , replay buffer  $\mathcal{D}$ , sampling size  $N_s$ , training epochs  $N_e$ , clipping
   factors  $\varepsilon$ , learning rates  $\alpha_\pi, \alpha_V$  and model parameters  $(\theta, \psi, \beta)$ .
3: for  $N_e$  epochs do
4:   Clear the replay buffer  $\mathcal{D}$ .
5:   // Data collection
6:   for  $N_s$  sampling steps do
7:     Randomly sample an instance  $x$  from  $\mathcal{X}$ .
8:     Run the MILP solver to optimize instance  $x$  with configuration policy  $\pi$ , collecting  $N_x$ 
       episodes of data  $\left\{ \left\{ (s_t^{(i)}, a_t^{(i)}, r_t^{(i)}) \right\}_{t=0}^{T(N_x)} \right\}_{i=1}^{N_x}$  from  $T(N_x)$  separation rounds at  $N_x$  nodes.
9:     Append collected episodes to  $\mathcal{D}$ .
10:  end for
11:  // Model Optimization via PPO
12:  Compute returns  $\hat{R}_1, \dots, \hat{R}_T$  and advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$  for each episode in  $\mathcal{D}$ .
13:  for each minibatch  $\mathcal{D}_b \subset \mathcal{D}$  do
14:    Compute ratio  $r_t(\theta) \leftarrow \mathbb{E}_{\mathcal{D}_b} \left\{ \frac{\pi(a_t|\Phi(s_t))}{\pi_{\text{old}}(a_t|\Phi(s_t))} \right\}$ .
15:    Compute actor loss  $L_{\text{actor}}(\theta) \leftarrow \mathbb{E}_{\mathcal{D}_b} \left\{ \min(r_t(\theta) \cdot \hat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) \cdot \hat{A}_t) \right\}$ .
16:    Update  $\theta \leftarrow \theta + \alpha_\pi \nabla_\theta L_{\text{actor}}(\theta)$ .
17:    Compute critic loss  $L_{\text{critic}}(\psi, \beta) \leftarrow \mathbb{E}_{\mathcal{D}_b} \left\{ (V(\Phi(s_t)) - \hat{R}_t)^2 \right\}$ .
18:    Update  $(\psi, \beta) \leftarrow (\psi, \beta) + \alpha_\beta \nabla_{(\psi, \beta)} L_{\text{critic}}(\psi, \beta)$ .
19:  end for
20: end for

```

Table 5: Hyperparameters used in DynSep.

Type	Parameter	Value
SCIP Solver	Number of separators K	22
	Frequency of each activated separators f_i	10
	Upper limit of separation rounds T	5
	time limit per instance	300
PPO in Alg. 1	Training epoch N_e	100
	Sampling size N_s per epoch	16
	Minibatch size $ \mathcal{D}_b $	
	MIPLIB mixed neos	8
	MIPLIB mixed supportcase	6
	Other benchmarks	16
	Clipping factor ε	0.2
	Optimizer	Adam
	learning rates α_π, α_V	0.0001
	learning rate decay for every k_{lr} step	5
	learning rate decay rate α_{lr}	0.96
Transformer	Embedding dimension of attention d	64
	Number of attention heads	4
	Number of attention layers	4
	Attention dropout	0.0
	Activation of FFN	GeLU
	Embedding dimension of FFN d_{FFN}	256
	Number of FFN layers	2
	Layer number of the linear actor head	1
	Layer number of the linear critic head	1

D.3 Hardware Specification

Training and evaluation on the easy and medium datasets were performed on a single machine equipped with eight GPUs (NVIDIA GeForce RTX 2080 Ti) and two Intel E5-2667 v4 CPUs (32 logical cores), while experiments on the hard datasets used a single machine equipped with eight GPUs (NVIDIA GeForce GTX 3090 Ti) and two Intel Gold 6246R CPUs (64 logical cores) for hard datasets.

D.4 Baselines

We provide additional implementation details for our baseline methods:

Search(ρ): This method randomly samples ρ configurations then applies one with the best performance on the validation set. The validation set is a subset of the training data, with a size equal to that of the corresponding test set for each MILP benchmark.

Prune: This method deactivates separators with no contribution during the evaluation on the validation set. Specifically, if a separator’s statistics—namely, Cut Application Rate, DomReds, and Cutoffs (as defined in Section B.3)—are all zero, the separator is deactivated. The validation set used here is partitioned identically to that in the Search(ρ) method.

L2Sep [6]: We configure the SCIP solver parameters in alignment with our default settings, except for the learned separator activation statuses. Specifically, we impose a 300-second time limit, set the frequency $f_i = 10$ for all activated separators, and enable both presolve and heuristic mechanisms. The validation set for L2Sep is partitioned identically to that used in the Search(ρ) method. We define the size of the restricted configuration space as 15 for easy datasets, 20 for medium datasets, and 25 for hard datasets.

LLM4Sep [7]: The LLM4Sep baseline utilizes the DeepSeek Chat API to generate separator configurations. The context provided to the language model includes detailed descriptions of each separator, as outlined in Section B.5, along with information regarding the problem structures the separators operate on and their computational characteristics. Additionally, the model receives a comprehensive description of the MILP problem, encompassing the general formulation and the summary text of MILP objectives and constraints.

E Additional Results

E.1 Motivation Results on effects of different configurations

Fig. 2(b) in our main text shows motivation results for five benchmarks, with alternative names of D1–Set Covering, D2–Max Independent Set (MIS), D3–MIK, D4–Load Balancing, D5–MIPLIB mixed neos. Here we provide more results for all nine benchmarks.

The left panel of Fig. 5 shows the best-performing configuration per instance identified by four randomization strategies applied to separator configurations. It extends the analysis from Fig. 2(b) in the main text by presenting results across nine benchmark datasets: D1–Set Covering, D2–Max Independent Set (MIS), D3–Multiple Knapsack, D4–CORLAT, D5–MIK, D6–Anonymous, D7–Load Balancing, D8–MIPLIB Mixed Neos, and D9–MIPLIB Mixed Supportcase. For each instance within these datasets, we evaluated 50 random configurations.

The right panel depicts the average performance across instances for each dataset. Our observations reveal substantial performance variance across all four strategies, underscoring the significant impact of the specific separator parameters and dynamic configurations on solver efficiency. Notably, the Priority strategy exhibits comparatively lower variance in performance. This is attributed to the fact that priority adjustments influence only the relative ordering of separators within a separation round; since the LP relaxation is not re-solved until the round concludes, such reordering has minimal effect on overall solver performance.

Furthermore, Fig. 6 illustrates the impact of varying the maximum number of separation rounds per node, denoted as r_{\max} , on solver performance across nine benchmark datasets. Each plot shows the

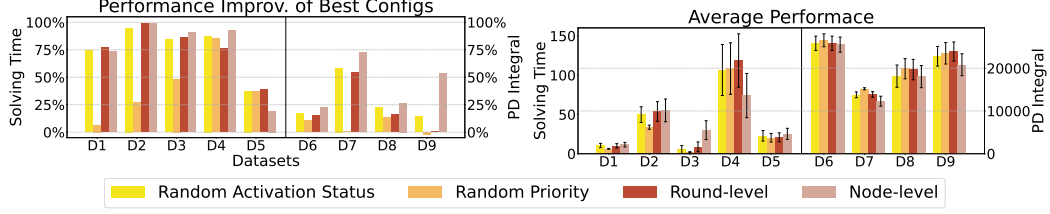


Figure 5: **Left:** Performance improvement of the best configurations found by different random strategies on nine benchmarks. The y-axis represents the relative improvement compared to the default setting. **Right:** Average performance of configurations sampled by different random strategies on nine benchmarks. The y-axis represents the real performance under two metrics. Specifically, Datasets D1–D5 use solving time (left) as the metric, while D6–D9 use PD integral (right). Each bar represents a specific strategy to get random configurations.

average solving time (red line, left y-axis) and PD integral (blue line, right y-axis) across all instances in the dataset. The two metrics exhibit highly consistent trends in each benchmark, indicating a strong correlation between solving time and PD integral. The results also show that changes in r_{\max} lead to significant performance variability; however, increasing r_{\max} does not universally enhance performance, and the optimal value of r_{\max} varies among datasets. Prior work [48] also observes that solver performance is sensitive to the maximum number of cut rounds and learns a data-driven stopping policy; however, it does not model per-round separator configuration, whereas we jointly decide when to halt and which separators to activate each round.

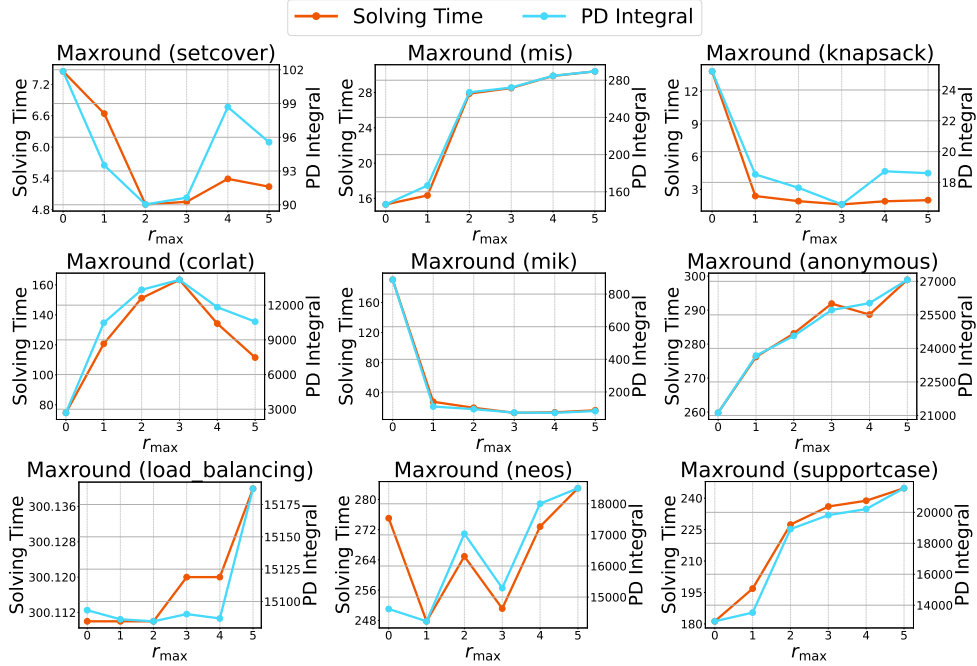


Figure 6: Effect of varying maximum round r_{\max} on solver performance for nine benchmarks. Each plot shows the average solving time (red line, left y-axis) and PD integral (blue line, right y-axis) across all instances in the dataset.

E.2 Evaluation Results on Other Metrics

We provide evaluation results of nine benchmarks for two other metrics of the primal-dual gap (PD gap) and total number of nodes (Nnodes) in Table 6. The PD gap reflects the solution quality achieved by the solver, while Nnodes indicates the size of the B&B search tree—an indirect measure of solving

Table 6: **Evaluation results on nine benchmarks about two other metrics of primal-dual gap (PD gap) and total number of nodes (Nnodes).** Best performance is in bold. The values report the mean (standard deviation) of time and PD integral metrics.

Easy: Set Covering			Easy: Max Independent Set		Easy: Multiple Knapsack	
Method	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓
NoCuts	0.0 (0.0)	114.84 (413.86)	0.0 (0.0)	529.31 (1703.82)	0.0 (0.0)	17847.64 (42453.94)
Default	0.0 (0.0)	1.11 (1.06)	0.0 (0.0)	1.0 (0.0)	0.0 (0.0)	36.91 (85.56)
Search(50)	0.0 (0.0)	1.16 (1.59)	0.0 (0.0)	123.41 (347.52)	0.0 (0.0)	3164.31 (4942.28)
Prune	0.0 (0.0)	207.61 (439.61)	0.0 (0.0)	403.72 (810.19)	0.0 (0.0)	12.19 (34.42)
L2Sep(R1)	0.0 (0.0)	211.27 (402.0)	0.0 (0.0)	386.28 (790.7)	0.0 (0.0)	10495.97 (19385.29)
L2Sep(R2)	0.0 (0.0)	211.81 (401.94)	0.0 (0.0)	384.32 (787.77)	0.0 (0.0)	12144.57 (19489.3)
LLM4Sepasel	0.0 (0.0)	179.04 (412.59)	0.0 (0.0)	39.12 (64.62)	0.0 (0.0)	1883.54 (2884.54)
DynSep (Ours)	0.0 (0.0)	1.0 (0.0)	0.0 (0.0)	1.0 (0.0)	0.0 (0.0)	5.2 (13.59)

Medium: CORLAT			Medium: MIK		Hard: Anonymous	
Method	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓
NoCuts	2.67e+18 (1.26e+19)	57516.67 (96475.09)	0.02 (0.03)	176742.17 (126999.33)	1.83e+19 (3.86e+19)	13034.68 (11248.92)
Default	2.73e+19 (4.46e+19)	304.77 (456.1)	0.0 (0.0)	5504.77 (6375.84)	5.5e+19 (4.96e+19)	1894.13 (4139.96)
Search(50)	4e+18 (1.96e+19)	35564.78 (68651.06)	0.0 (0.0)	13134.6 (11543.03)	4e+19 (4.90e+19)	4602.75 (6696.04)
Prune	2e+18 (1.4e+19)	99579.13 (141189.24)	0.09 (0.02)	532615.47 (169443.58)	6.67e+18 (2.45e+19)	22442.38 (14908.91)
L2Sep(R1)	4e+18 (1.98e+19)	84423.58 (117122.52)	0.0 (0.0)	5157.9 (6471.25)	1.88 (2.19)	15856.25 (14404.64)
L2Sep(R2)	2e+18 (1.41e+19)	81853.7 (116670.12)	0.0 (0.0)	3616.2 (3105.22)	5e+18 (2.24e+19)	16151.85 (14903.3)
LLM4Sepasel	2e+18 (1.41e+19)	38808.98 (71956.13)	0.0 (0.0)	5653.0 (6682.38)	6e+19 (5.03e+19)	3835.15 (7788.15)
DynSep (Ours)	0.0 (0.0)	4084.16 (11015.44)	0.0 (0.0)	3076.0 (2666.61)	1.99 (2.96)	15011.2 (8355.63)

Hard: Load Balancing			Hard: MIPLIB mixed neos		Hard: MIPLIB mixed supportcase	
Method	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓	PD gap ↓	Nnodes ↓
NoCuts	0.97 (0.12)	1.0 (0.0)	2.5e+19 (4.33e+19)	148834.67 (93798.58)	10.94 (25.26)	2204.17 (3130.9)
Default	0.97 (0.12)	1.0 (0.0)	2.5e+19 (4.33e+19)	12927.5 (16777.1)	2.5e+19 (4.33e+19)	22.25 (54.64)
Search(50)	0.09 (0.01)	10.24 (12.67)	2.5e+19 (4.33e+19)	35011.0 (44832.71)	0.1 (0.26)	482.12 (729.38)
Prune	0.49 (0.05)	150.51 (65.02)	2.5e+19 (4.33e+19)	202703.25 (149382.93)	12.51 (26.94)	6162.54 (13362.48)
L2Sep(R1)	0.59 (0.37)	86.36 (55.62)	2.5e+19 (5e+19)	66880.0 (103828.32)	12.45 (23.27)	1622.38 (3290.69)
L2Sep(R2)	0.59 (0.37)	85.36 (55.23)	2.5e+19 (5e+19)	120425.0 (119640.78)	9.29 (17.1)	3414.75 (6492.98)
LLM4Sepasel	0.12 (0.03)	20.73 (16.85)	2.5e+19 (5e+19)	102753.75 (75635.08)	2.5e+19 (4.63e+19)	37.38 (102.48)
DynSep (Ours)	0.09 (0.02)	9.99 (11.41)	2.5e+19 (4.33e+19)	119616.25 (105217.02)	7.86 (20.31)	69.38 (131.34)

effort, though a smaller tree does not necessarily imply faster solving. The results show that DynSep consistently solves all easy and medium instances to optimality, achieving an average PD gap of zero. Notably, DynSep is the only configuration method that solves all instances to optimality with zero primal-dual gaps, highlighting the effectiveness of DynSep for fine-grained separator configurations.

E.3 Evaluation on Additional MIPLIB Datasets

We have extended our evaluation beyond MIPLIB mixed neos and mixed supportcase, including two real-world datasets from the Distributional MIPLIB benchmark [49]:

- Maritime Inventory Routing Problem (MIRP). MIRP arises in bulk shipping logistics, integrating vessel routing and port inventory decisions under capacity and inventory constraints. Typical instances of MIRP feature on an average of 15080 binary variables, 19576 continuous variables, and 44430 constraints.
- Seismic-Resilient Pipe Network Planning (SRPN). SRPN involves optimizing municipal water pipe network design to ensure resilience under seismic disturbances, targeting service continuity to critical facilities (e.g. hospitals) while minimizing upgrade or restoration costs within budget. Typical instances of SRPN feature on an average of 3016 binary variables, 3016 continuous variables, and 5917 constraints.

The results are summarized in Table 7. For each dataset, we report solving time (Time), primal–dual gap integral (PD integral), and primal–dual gap (PD gap). All three metrics are lower-is-better, where Time and PD integral reflect solver efficiency and convergence speed, and PD gap quantifies how close the solver comes to the optimal. We set the time limit as 600 seconds for each instance. These results show that DynSep delivers marked performance gains on additional real-world datasets (MIRP and SRPN). Compared to the other configuration methods, DynSep significantly improves

Table 7: Evaluation results on MIPLIB MIRP & SRPN Benchmarks, with 600-second time limit.

Hard: MIPLIB MIRP ($n = 34656$, $m = 44430$)				Hard: MIPLIB SRPN ($n = 6032$, $m = 5917$)		
Method	Time(s) ↓	PD integral ↓	PD gap ↓	Time(s) ↓	PD integral ↓	PD gap ↓
NoCuts	486.68 (198.71)	34735.55 (19014.81)	6.67e+18 (1.92e+19)	280.7 (277.17)	11020.69 (13133.86)	0.28 (0.42)
Default	580.98 (61.65)	52362.46 (13476.17)	5.38e+19 (4.97e+19)	332.04 (271.0)	11687.01 (11993.95)	0.21 (0.31)
Search(20)	492.85 (190.1)	33028.81 (16299.41)	6.30e+18 (2.11e+19)	384.02 (273.02)	14571.56 (12206.64)	0.26 (0.28)
Prune	501.49 (174.33)	35542.82 (18572.11)	8.33e+18 (2.19e+19)	296.08 (277.77)	9211.76 (10664.7)	0.19 (0.26)
LLM4Sepasel	511.97 (161.03)	34573.52 (18681.26)	5e+18 (2.24e+19)	300.66 (284.75)	8421.65 (10209.65)	0.14 (0.22)
DynSep (Ours)	482.13 (205.78)	30838.39 (17377.29)	1.38 (1.41)	294.77 (274.22)	7581.16 (8814.39)	0.1 (0.17)

Table 8: Comparison between default setting and our method (DynSep) on all 235 MIPLIB 2017 instances.

Hard: MIPLIB 2017		
Method	Time(s) ↓	PD integral ↓
Default	258.77 (93.22)	17153.69 (12674.03)
DynSep (Ours)	238.88 (107.33)	15092.16 (12719.67)

both convergence speed (as demonstrated by reduced PD integral) and solution quality (evidenced by lower PD gap).

Furthermore, we have tested our method on the full set of 240 MIPLIB 2017 benchmark instances. The results in Table 8 show that DynSep delivers notable improvements in solving efficiency in the challenging MIPLIB 2017 dataset.

Specifically, we set the time limit as 300 seconds and excluded five instances whose presolving time exceeded 300 seconds: *neos-3402454-bohle*, *neos-4722843-widden*, *mzzv42z*, *neos-5052403-cygnnet*, *proteindesign121hz512p9*, and *proteindesign122trx11p8*, which is a common removing criterion for MIPLIB2017 benchmark [18, 42]. The remaining 235 instances were split into a 70% training set and a 30% test set. Table 8 reports the overall average performance of our method across all 235 instances. These experiments confirm that our approach delivers notable improvements in solving efficiency, even when evaluated on the more challenging benchmark set.

E.4 Extended DynSep to Broader Solver Hyperparameters

Our proposed DynSep framework is inherently extensible to a broader set of solver hyperparameters beyond separator activation and timing. This is achieved by adapting the policy network: we model the outputs of additional parameters as a logistic-normal distribution, followed by optional discretization to support both integer and continuous parameters. This enables flexible, differentiable control of arbitrary solver parameters.

To substantiate the above claim, we conducted additional experiments on three critical hyperparameter groups as follows, while retaining the original tuning mechanism for separator activation status (+1, 0, -1) and round termination (m_t).

- **para group 1: Cut Depth / Aggressiveness (sepastore/age in SCIP).** Controlled by solver parameters `separating/cutagelimit` and `separating/poolfreq`.
- **para group 2: Cut selection thresholds (e.g., efficacy vs. orthogonality).** Controlled by solver parameters `separating/minefficacy` and `cutselection/hybrid/minortho`.
- **para group 3: Separation frequency per node.** Controlled by solver parameters `separating/cutagelimit` and `separating/poolfreq`.

We provide the experimental results in Table 9. We evaluated our method on three benchmark datasets: Multiple Knapsack, Corlat, and MIPLIB mixed supportcase, measuring solving time (Time), primal-dual gap integral (PD integral), and primal-dual gap (PD gap). All three metrics are lower-is-better, where Time and PD integral reflect solver efficiency and convergence speed, and PD gap quantifies how close the solver comes to the optimal. We compared our original DynSep method with its extended versions incorporating three additional parameter groups. Results show

Table 9: Experiments on three extended hyperparameter groups configured by DynSep.

Method	Easy: Multiple Knapsack ($n = 720$, $m = 72$)			Medium: Corlat ($n = 466$, $m = 486$)			Hard: MIPLIB mixed supportcase ($n = 19766$, $m = 19910$)		
	Time(s) ↓	PD integral ↓	PD gap ↓	Time(s) ↓	PD integral ↓	PD gap ↓	Time(s) ↓	PD integral ↓	PD gap ↓
DynSep (Ours)	0.52 (0.24)	9.71 (5.39)	0.0 (0.0)	22.96 (38.93)	2233.42 (3868.43)	0.0 (0.0)	132.50 (130.32)	9212.24 (9840.56)	7.86 (20.31)
Para Group 1	0.65 (0.59)	9.01 (4.59)	0.0 (0.0)	47.36 (70.18)	4580.9 (7028.59)	4e+18 (1.96e+19)	141.54 (124.79)	8610.24 (8190.9)	0.17 (0.28)
Para Group 2	0.36 (0.23)	7.91 (4.51)	0.0 (0.0)	42.17 (79.8)	1971.18 (3704.0)	0.01 (0.03)	167.64 (134.41)	8661.57 (7499.68)	0.16 (0.29)
Para Group 3	0.65 (0.23)	10.03 (5.31)	0.0 (0.0)	24.71 (54.74)	1922.77 (3826.2)	0.0 (0.02)	141.09 (125.39)	8362.53 (6834.04)	0.17 (0.28)

Table 10: Execution Time for each decision step of DynSep to configure separators

	Set Covering	MIS	Knapsack	CORLAT	MIK	Anonymous	Load Balancing	Neos	Supportcase
Avg. Latency (s)	0.33	0.22	0.09	0.05	0.31	0.41	3.04	0.11	0.39
Max. Latency (s)	1.09	1.01	0.71	0.70	1.07	1.94	4.51	2.02	6.85

that across all datasets, DynSep and its extended variants consistently outperform the default solver configuration reported in the main paper. Furthermore, the differences among the parameter groups are relatively small, yet certain combinations (e.g., Para Group 2 on Knapsack and Para Group 3 on mixed supportcase) yield further performance improvements in both Time and PD integral. These results validate that DynSep can flexibly extend to control a broader set of solver hyperparameters. Furthermore, carefully chosen parameter combinations can yield additional gains in solving speed and convergence.

E.5 Overhead Evaluation

E.5.1 Latency of Policy Inference

We have provided per-decision latency (the latency of policy inference per decision step) and the total inference time through the solving process as follows.

Per-decision latency. Table 10 reports the average ("Avg. latency") and worst-case ("Max. latency") time taken for a single policy call across the entire branch-and-cut process. These values reflect the inference latency introduced by DynSep policy for each configuration decision.

Our results reveal that per-decision latency increases as instance size grows. On small to medium-sized datasets, the worst-case latency remains around 1 second per policy call. For the larger and more complex problem instances (with tens of thousands of variables and constraints), the worst-case latency ranges from 1 to 7 seconds per decision. In contrast, the average inference latency stays below 0.5 seconds across all datasets, with the exception of the largest load balancing instance (approximately 61K variables, 64K constraints), where the average latency rises to 3 seconds.

Table 11 provides the total inference time ("Infer. Time") over different datasets and solver time limits, along with the inference overhead rate ("Overhead Rate"), which represents the percentage of policy inference for the total solving time ("Sol. Time"). Table 11 shows that DynSep incurs a modest configuration overhead, contributing a negligible fraction of the total solving time even on large-scale instances. While the configuration time tends to increase with problem size—e.g., from under one second on small benchmarks to several tens of seconds on the largest ones—it remains practically affordable relative to the performance gains achieved. Nonetheless, there is potential to further optimize this step, and future work may explore more lightweight models to reduce configuration latency without compromising effectiveness.

Notably, our configuration policy is encapsulated within a custom SCIP separator. Thus, we report the total inference time via SCIP’s built-in `SCIPsepaGetTime()` function. In contrast, the per-decision latency values are logged using wall-clock timing at each policy call, including overhead from time recording and logging. Consequently, the sum of the per-decision latencies naturally exceeds the total inference time, since the latter excludes the additional overhead introduced by frequent timing operations.

E.5.2 Memory Overhead

We analyze the memory inference overhead as follows.

Per-decision memory overhead: Table 12 below show the peak GPU memory usage per policy call.

Table 11: Inference Time for DynSep to configure separators during solving process. Three blocks: nine datasets in our manuscript (300-second time limit), four hard datasets (3600-second time limit), and MIPLIB MIRP & SRPN (600-second time limit).

	Set Covering	MIS	Knapsack	CORLAT	MIK	Anonymous	Load Balancing	Neos	Supportcase
Infer. Time (s)	1.05 (0.34)	0.41 (0.23)	0.42 (0.51)	16.56 (21.07)	1.57 (0.55)	14.73 (20.58)	10.54 (0.73)	23.9 (27.99)	14.16 (19.65)
Sol. Time (s)	1.51 (0.27)	0.53 (0.20)	0.52 (0.24)	22.96 (38.93)	10.99 (9.44)	241.89 (100.75)	300.04 (0.08)	235.19 (112.26)	132.50 (130.32)
Overhead Rate (%)	69.54	77.36	80.77	72.13	14.29	6.09	3.51	10.16	10.69

	Anonymous	Load Balancing	MIPLIB mixed neos	MIPLIB mixed supportcase
Infer. Time (s)	21.32 (16.64)	16.01 (1.57)	86.96 (114.55)	12.31 (9.9)
Sol. Time (s)	2397.95 (1551.2)	3600.04 (0.07)	2724.85 (1515.82)	567.82 (1154.86)
Overhead Rate (%)	0.89	0.44	3.19	2.17

	MIPLIB MIRP	MIPLIB SRPN
Infer. Time (s)	32.77 (29.27)	5.06 (2.28)
Sol. Time (s)	482.13 (205.78)	294.77 (274.22)
Overhead Rate (%)	6.80	1.72

Table 12: Per-decision memory overhead.

	Set Covering	MIS	Knapsack	CORLAT	MIK	Anonymous	Load Balancing	Neos	Supportcase
Avg. CPU mem.(MB)	2902.22	2804.20	2807.59	2786.58	2775.91	3153.15	3811.82	2824.13	2870.90
Max. CPU mem. (MB)	3081.70	2877.39	2889.04	2892.15	2877.77	3930.07	4852.27	3100.90	3688.11
Avg. GPU mem. (MB)	2.09	2.09	2.10	2.09	2.10	2.09	2.10	2.09	2.09
Max. GPU mem. (MB)	2.10	2.10	2.11	2.11	2.11	2.11	2.11	2.11	2.11

peak memory overhead for overall inference: The memory footprint required to store the encoder and policy model weights is 2.08 MB for each dataset. We track the peak GPU memories during the reference via the tool `torch.cuda.max_memory_allocated()`, which are list in Table 13.

E.6 Ablation Study

E.6.1 Module Ablation Analysis on Other Six Benchmarks

We evaluate DynSep and its ablated variants on other six benchmark datasets using solve time and the primal-dual (PD) gap integral as performance metrics. Table 14 summarizes the results. Specifically, the ablation study shows that while certain DynSep variants (e.g., w/o DynG&TF in Set Covering, w/o TF in MIS, w/o MaxR in Anonymous and Load Balancing) can slightly beat the full model on individual metrics for particular datasets, the full DynSep method consistently delivers robust, near-best performance overall. Overall, each component’s removal yields trade-offs, but the full DynSep model demonstrates consistently balanced performance across tasks.

E.6.2 Encoder Architecture Ablation

Table 15 shows that replacing the encoder’s GCN with a custom bipartite graph transformer (GT): a multi-head TransformerConv for edge-aware message passing, followed by residual-connected LayerNorm and a two-layer feed-forward block. *GT-encoder* shows no consistent improvement over *DynSep*, which may be due to increased inference overhead or the need for finer stability/tuning to realize gains from the transformer-style aggregation.

E.6.3 Hyperparameter Sensitivity Analysis

As shown in Table 16, we conducted robustness ablations over separator *frequency* (1, 5, 10, 20) and *maximum separation rounds* (3, 5, 10, 20) on three benchmarks (MIK, Corlat, and MIPLIB mixed Neos). Overall, across almost all tested settings, DynSep outperforms the default configuration, showing that the approach is reasonably stable to these hyperparameters. Below is our key observations.

Frequency. Moderate frequency (e.g., 5&10) gives the better trade-off. That is, small frequency causes separators to fire excessively, incurring high cut-generation overhead, whereas large frequency reduces opportunities for timely dual-bound tightening.

Table 13: Peak memory overhead for overall inference.

	Set Covering	MIS	Knapsack	CORLAT	MIK	Anonymous	Load Balancing	Neos	Supportcase
Peak GPU mem. (MB)	35.56	57.85	23.11	15.68	37.41	157.17	257.60	28.85	122.28

Table 14: Ablation results on other six benchmarks

Easy: Set Covering ($n = 1000, m = 500$)				Easy: Max Independent Set ($n = 500, m = 1953$)			Medium: MIK ($n = 413, m = 346$)		
Method	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓
NoCuts	7.45 (5.87)	NA	101.86 (55.59)	15.32 (5.82)	NA	146.4 (56.99)	190.28 (113.97)	NA	887.85 (859.76)
Default	5.24 (1.79)	29.66	95.56 (36.86)	30.4 (8.02)	-98.43	289.51 (103.81)	16.65 (18.06)	91.25	82.80 (56.24)
w/o MaxR	1.32 (0.72)	82.28	31.35 (11.32)	0.57 (0.24)	96.28	10.46 (2.83)	12.46 (8.81)	93.45	128.91 (67.89)
w/o TF	1.48 (0.35)	80.13	32.86 (6.25)	0.44 (0.09)	97.13	9.16 (1.77)	14.04 (12.99)	92.62	106.14 (65.99)
w/o DynG	1.25 (0.68)	83.22	29.63 (10.34)	0.56 (0.22)	96.34	10.19 (2.68)	11.88 (9.44)	93.76	116.16 (40.64)
w/o DynG&TF	1.23 (0.69)	83.49	29.33 (10.42)	0.55 (0.18)	96.41	9.97 (2.28)	12.43 (10.37)	93.47	127.63 (46.03)
DynSep (Ours)	1.51 (0.27)	79.73	33.88 (9.34)	0.53 (0.20)	96.54	9.66 (2.40)	10.99 (9.44)	94.22	134.15 (44.21)

Hard: Anonymous ($n = 37881, m = 49603$)				Hard: Load Balancing ($n = 61000, m = 64304$)			Hard: MIPLIB mixed supportcase ($n = 19766, m = 19910$)		
Method	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)
NoCuts	259.77 (75.71)	21117.12 (9234.01)	NA	300.11 (0.02)	15093.26 (940.68)	NA	181.26 (120.25)	12959.99 (10506.47)	NA
Default	298.92 (4.09)	27069.58 (4892.8)	-28.19	300.14 (0.02)	15187.19 (936.38)	-0.62	244.75 (105.8)	21561.09 (10434.42)	-66.37
w/o MaxR	243.92 (97.55)	14452.24 (9840.56)	31.56	300.13 (0.37)	3252.99 (454.96)	78.45	167.52 (112.43)	10158.06 (9568.77)	21.62
w/o TF	251.16 (90.02)	16238.64 (9292.53)	23.10	300.02 (0.03)	3740.29 (473.88)	75.22	143.54 (123.86)	10253.13 (9952.05)	20.89
w/o DynG	256.66 (77.74)	16903.77 (8941.95)	19.95	300.1 (0.02)	15020.22 (941.63)	0.48	145.76 (121.48)	11882.87 (11695.35)	8.31
w/o DynG&TF	246.48 (93.01)	18914.82 (9388.91)	10.43	300.04 (0.04)	3923.57 (539.21)	74.00	131.72 (130.92)	11369.53 (12085.07)	12.27
DynSep (Ours)	241.89 (100.75)	15656.7 (8996.14)	25.86	300.04 (0.08)	3720.26 (499.37)	75.35	132.50 (130.32)	9212.24 (9840.56)	28.92

Maximum Separation Round. Setting this value too low produces weak cuts and degrades performance, while setting it excessively high increases the computational cost of cut generation. Although this parameter shows some dataset sensitivity, MaxRound=5 is empirically near-optimal in our tests.

E.7 Generalization Study

We investigate complementary generalization performance of our method under two more settings:

- (1) **Cross-Domain Generalization Test:** training on one benchmark and testing on a dataset from a different domain;
- (2) **General-to-Specific Generalization Test:** training a general model on a mixed-category dataset (e.g. MIPLIB) and then evaluating on a specific class dataset.

E.7.1 Cross-Domain Generalization Test

To evaluate the across-domain generalization, we train our policy on one problem family and apply the learned policy to unseen problem families. Specifically, we train four separate DynSep policies (on Setcover, Knapsack, MIK, and Supportcase) and evaluate each of them across all nine benchmark families used in our manuscript. The table below lists the results, where we report solving time for easy and medium datasets, while additionally report primal–dual integral (PD integral) for hard datasets.

As shown in Table 17, policies trained on one problem type yield improvements over SCIP’s default in most unseen benchmarks, indicating effective transfer of our learned configuration strategy across NP-hard families.

E.7.2 General-to-Specific Generalization Test

We select 168 diverse instances from the MIPLIB 2017 benchmark [14] as our training set and learn a separator configuration policy on these instances. Notably, because the MIPLIB 2017 instances cover a wide variety of problem types and mixed-scenario structures, this learned policy could serve as a general configuration model. We then evaluate it—without any additional tuning—on four unseen, domain-specific datasets: Corlat, Load Balancing (LB), Maritime Inventory Routing Problem (MIRP), and Seismic-Resilient Pipe Network Planning (SRPN). We have extended our evaluation beyond MIPLIB mixed neos and mixed supportcase, including two real-world datasets from the Distributional MIPLIB benchmark [49]:

Table 15: Comparative results of different encoder architectures for DynSep on four datasets.

Easy: Multiple Knapsack ($n = 720$, $m = 72$)				Medium: Corlat ($n = 466$, $m = 486$)		
Method	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓
GT-encoder	0.71 (0.39)	94.87	11.02 (5.41)	46.26 (71.41)	38.04	4563.26 (7125.57)
DynSep (Ours)	0.52 (0.24)	96.24	9.71 (5.39)	22.96 (38.93)	69.25	2233.42 (3868.43)

Hard: MIPLIB mixed neos ($n = 6958$, $m = 5660$)				Hard: MIPLIB mixed supportcase ($n = 19766$, $m = 19910$)		
Method	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)
GT-encoder	243.52 (97.82)	12134.57 (12142.03)	16.99	150.38 (123.13)	9157.61 (9623.27)	29.34
DynSep (Ours)	235.19 (112.26)	8511.58 (12413.9)	41.78	132.50 (130.32)	9212.24 (9840.56)	28.92

Table 16: Sensitivity Analysis of hyperparameters *Frequency* and *Maximum Separation Round* on three datasets.

Medium: MIK ($n = 413$, $m = 346$)				Medium: Corlat ($n = 466$, $m = 486$)			Hard: MIPLIB mixed neos ($n = 6958$, $m = 5660$)		
Method	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)
NoCuts	190.28 (113.97)	NA	887.85 (859.76)	74.66 (122.23)	NA	2687.68 (6209.48)	275.04 (43.23)	14618.53 (12214.63)	NA
Default	16.65 (18.06)	91.25	82.80 (56.24)	111.55 (132.19)	-49.41	10573.14	282.98 (29.49)	18500.5 (9386.15)	-26.56
Freq=1	12.74 (9.89)	93.30	124.22 (43.24)	53.77 (84.77)	27.98	4582.15 (7398.26)	252.12 (83.27)	8756.3 (12305.23)	40.10
Freq=5	13.04 (11.04)	93.15	86.55 (38.54)	38.44 (49.72)	48.51	3513.13 (4533.24)	243.91 (97.16)	9248.52 (12053.99)	36.73
Freq=10 (Ours)	10.99 (9.44)	94.22	134.15 (44.21)	22.96 (38.93)	69.25	2233.42 (3868.43)	235.19 (112.26)	8511.58 (12413.9)	41.78
Freq=20	12.83 (9.6)	93.26	213.15 (124.09)	49.14 (72.87)	34.18	4658.09 (7252.41)	261.86 (66.13)	8789.54 (12249.16)	39.87

Medium: MIK ($n = 413$, $m = 346$)				Medium: Corlat ($n = 466$, $m = 486$)			Hard: MIPLIB mixed neos ($n = 6958$, $m = 5660$)		
Method	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	Improv. ↑ (time, %)	PD integral ↓	Time(s) ↓	PD integral ↓	Improv. ↑ (PD Int., %)
NoCuts	190.28 (113.97)	NA	887.85 (859.76)	74.66 (122.23)	NA	2687.68 (6209.48)	275.04 (43.23)	14618.53 (12214.63)	NA
Default	16.65 (18.06)	91.25	82.80 (56.24)	111.55 (132.19)	-49.41	10573.14	282.98 (29.49)	18500.5 (9386.15)	-26.56
MaxRound=3	13.63 (12.04)	92.84	150.19 (46.77)	107.26 (124.27)	-43.66	8965.26 (12401.06)	239.22 (105.28)	8581.5 (12378.07)	41.30
MaxRound=5 (Ours)	10.99 (9.44)	94.22	134.15 (44.21)	22.96 (38.93)	69.25	2233.42 (3868.43)	235.19 (112.26)	8511.58 (12413.9)	41.78
MaxRound=10	17.6 (12.68)	90.75	182.71 (121.39)	40.45 (68.93)	45.82	3717.64 (6764.77)	249.41 (87.63)	8846.91 (12224.34)	39.48
MaxRound=20	16.2 (11.25)	91.49	169.8 (71.61)	36.71 (55.75)	50.83	3609.11 (5562.51)	252.59 (82.11)	10945.82 (11486.55)	25.12

As shown in Table 18, the general configuration model consistently outperforms the solver’s default settings in both solve time and convergence behavior, demonstrating good generalizability of our method.

E.8 Visualization of Separator Configurations on Nine Benchmarks

We provide visualization of separator configurations on nine benchmarks in Figs. 7- 15. Figs. 7 and 8 show that our learned policy uniformly reduces the maximum number of separation rounds to $r_{\max} = 3$ for easy benchmarks, Set Covering and MIS, demonstrating that our learned decision on maximum rounds effectively prunes unnecessary cutting rounds on simple problems. The heatmap reveals that the separator configuration is not static but varies dynamically across separation rounds (shown along the y-axis), suggesting that the model is timing the application of various separators to coincide with the stage of cut generation. Furthermore, the fact that learned activation values are not restricted to $\{-1, 0, 1\}$ but take intermediate real values indicates the policy differentiates between individual instances (and even between nodes) when selecting separators. In other words, it has learned a nuanced, instance-wise (and node-wise) cutting strategy rather than a one-size-fits-all rule.

Table 17: Cross-domain generalization on nine benchmarks. Policies are trained on one problem family and evaluated on unseen families.

Easy: Set Covering ($n = 1000, m = 500$)			Easy: Max Independent Set ($n = 500, m = 1953$)		Easy: Multiple Knapsack ($n = 720, m = 72$)	
Method	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓
Default	5.24 (1.79)	95.56 (36.86)	30.4 (8.02)	289.51 (103.81)	2.01 (1.82)	18.6 (10.49)
Train on Setcover	NA	NA	1.0 (1.38)	13.57 (9.99)	0.68 (0.42)	10.73 (6.12)
Train on Knapsack	2.02 (0.62)	40.96 (9.93)	0.76 (0.29)	12.37 (3.53)	NA	NA
Train on MIK	7.74 (4.33)	80.78 (40.05)	5.21 (3.38)	29.81 (19.2)	1.2 (1.09)	11.91 (5.52)
Train on supportcase	2.21 (0.59)	43.95 (9.35)	0.67 (0.3)	11.72 (3.63)	0.78 (1.06)	10.65 (5.97)

Medium: Corlat ($n = 466, m = 486$)			Medium: MIK ($n = 413, m = 346$)		Hard: Anonymous ($n = 37881, m = 49603$)	
Method	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓
Default	111.55 (132.19)	10573.14 (13070.46)	16.65 (18.06)	82.80 (56.24)	298.92 (4.09)	27069.58 (4892.8)
Train on Setcover	43.79 (76.54)	4042.34 (7652.31)	24.8 (21.41)	139.1 (45.45)	250.46 (87.15)	19701.65 (9639.56)
Train on Knapsack	27.56 (58.48)	2485.67 (5777.55)	12.48 (10.29)	139.08 (44.39)	253.74 (80.42)	19183.3 (8941.1)
Train on MIK	34.54 (69.18)	2163.38 (4884.15)	NA	NA	268.43 (56.97)	20715.25 (8615.27)
Train on supportcase	20.94 (50.99)	2016.37 (5095.97)	163.6 (98.18)	785.46 (577.89)	256.33 (79.04)	17922.89 (8384.44)

Hard: Load Balancing ($n = 61000, m = 64304$)			Hard: MIPLIB mixed neos ($n = 6958, m = 5660$)		Hard: MIPLIB mixed supportcase ($n = 19766, m = 19910$)	
Method	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓
Default	300.14 (0.02)	15187.19 (936.38)	282.98 (29.49)	18500.5 (9386.15)	244.75 (105.8)	21561.09 (10434.42)
Train on Setcover	300.05 (0.05)	4411.86 (514.13)	258.22 (72.37)	13512.27 (12604.76)	141.31 (125.51)	12171.62 (11385.67)
Train on Knapsack	300.04 (0.05)	4583.45 (554.52)	256.34 (75.63)	13343.38 (12514.04)	166.9 (116.16)	13216.93 (11281.29)
Train on MIK	300.04 (0.05)	5559.96 (1336.87)	249.83 (86.89)	13366.13 (12606.74)	187.09 (127.76)	11583.1 (9504.89)
Train on supportcase	300.09 (0.31)	9421.93 (665.37)	246.91 (91.95)	13639.73 (12439.81)	NA	NA

Table 18: Generalization performance of our DynSep model trained on MIPLIB 2017, evaluated on four unseen MILP scenarios. (300-second time limit for Corlat & LB; 600-second for MIPRP & SRPN)

Medium: Corlat ($n = 466, m = 486$)		Hard: Load Balancing ($n = 61000, m = 64304$)		
Method	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓
Default 2.01 (1.82)	5.24 (1.79) 18.6 (10.49)	95.56 (36.86)	30.4 (8.02)	289.51 (103.81)
DynSep (ours) trained on MIPLIB 2017	46.05	4289.33	300.08	4792.71
Hard: MIPLIB MIRP ($n = 34656, m = 44430$)		Hard: MIPLIB SRPN ($n = 6032, m = 5917$)		
Method	Time(s) ↓	PD integral ↓	Time(s) ↓	PD integral ↓
Default	580.98 (61.65)	52362.46 (13476.17)	332.04 (271.0)	11687.01 (11993.95)
DynSep (ours) trained on MIPLIB 2017	487.59	33193.25	288.48	7582.60

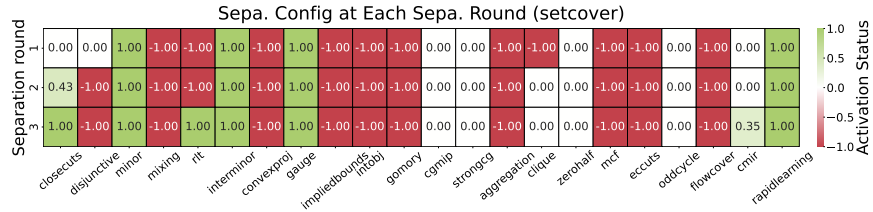


Figure 7: Separator configs at each separation round of Set Covering benchmark.

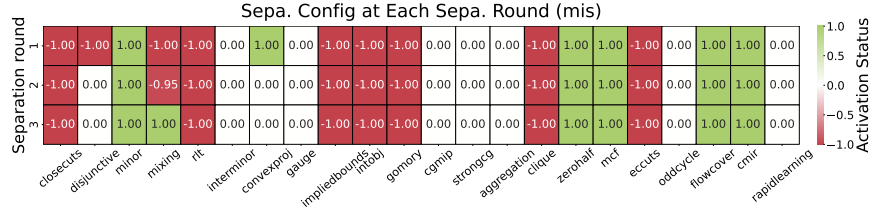


Figure 8: Separator configs at each separation round for Maximum Independent Set benchmark.

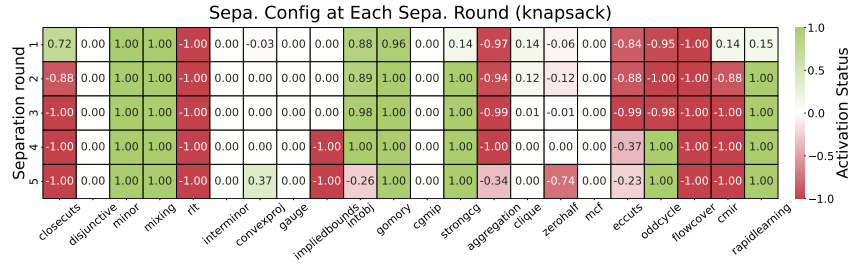


Figure 9: Separator configs at each separation round for Multiple Knapsack benchmark.

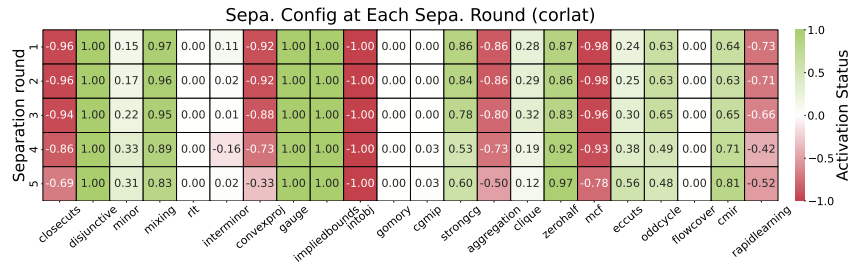


Figure 10: Separator configs at each separation round for CORLAT benchmark.

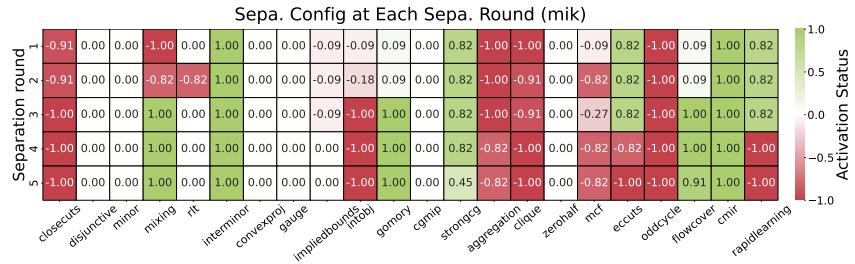


Figure 11: Separator configs at each separation round for MIK benchmark.

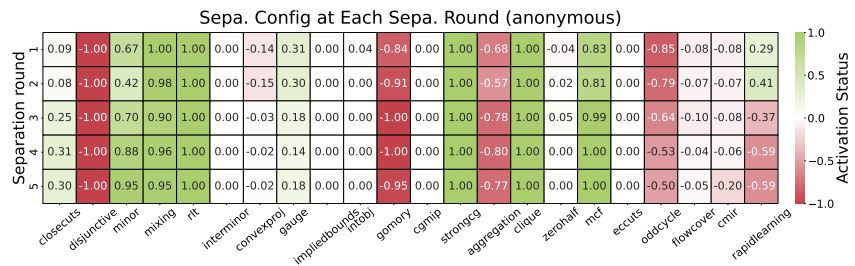


Figure 12: Separator configs at each separation round for Anonymous benchmark.

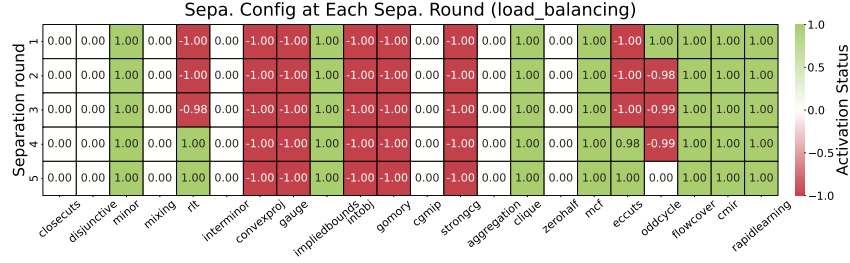


Figure 13: Separator configs at each separation round for Load Balancing benchmark.

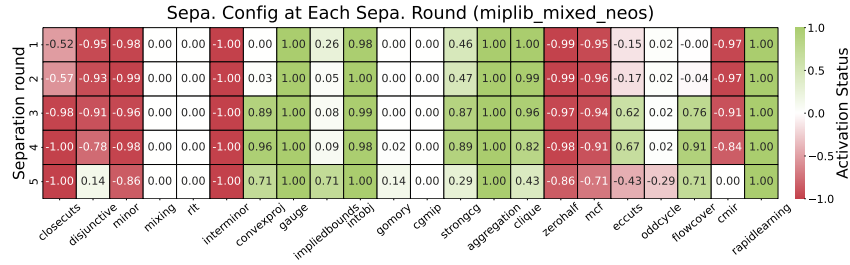


Figure 14: Separator configs at each separation round for MIPLIB mixed neos benchmark.

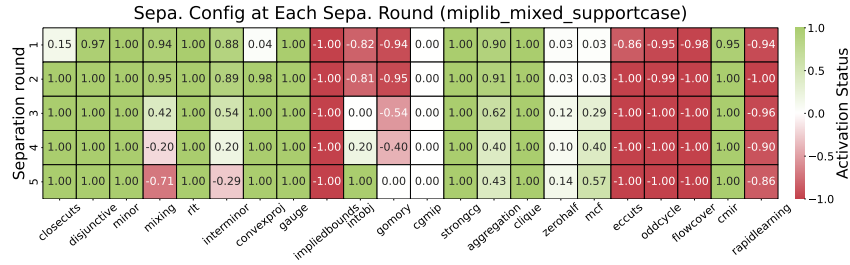


Figure 15: Separator configs at each separation round for MIPLIB mixed supportcase benchmark.