

Model solving

Ellen Vitercik

ML to speed up MILP solvers

- Modern MILP solvers expose **hundreds** of parameters

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETTILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 160	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 20	CPX_PARAM_FPHEUR 72
CPX_PARAM_NODESET 103				FRACCAND 73
CPX_PARAM_NODESET2 104				FRACCUTS 73
CPX_PARAM_NODESET3 105				FRACPASS 74
CPX_PARAM_NODESET4 106				GUBCOVERS 75
CPX_PARAM_NODESET5 107				HEURFREQ 76
CPX_PARAM_NODESET6 108				IMPLBD 76
CPX_PARAM_NODESET7 109				INTSOLFILEPREFIX 78
CPX_PARAM_NODESET8 110				INTSOLLIM 79
CPX_PARAM_NODESET9 111				ITLIM 80
CPX_PARAM_NODESET10 112				LANDPCUTS 82
CPX_PARAM_NODESET11 113				LBHEUR 81
CPX_PARAM_NODESET12 114				LPMETHOD 136
CPX_PARAM_NODESET13 115				MCFCUTS 82
CPX_PARAM_NODESET14 116				MEMORYEMPHASIS 83
CPX_PARAM_NODESET15 117				MIPCBREDLP 84
CPX_PARAM_NODESET16 118				MIPDISPLAY 85
CPX_PARAM_NODESET17 119				MIPEMPHASIS 87
CPX_PARAM_NODESET18 120				MIPINTERVAL 88
CPX_PARAM_NODESET19 121				MIPKAPPASTATS 89
CPX_PARAM_NODESET20 122				MIPORDIND 90
CPX_PARAM_NODESET21 123				MIPORDTYPE 91
CPX_PARAM_NODESET22 124				CPX_PARAM_MIPSEARCH 92
CPX_PARAM_NODESET23 125				CPX_PARAM_MIQCPSTRAT 93
CPX_PARAM_NODESET24 126				CPX_PARAM_MIRCUTS 94
CPX_PARAM_NODESET25 127				CPX_PARAM_MPSSLONGNUM 94
CPX_PARAM_NODESET26 128				CPX_PARAM_NETDISPLAY 95
CPX_PARAM_NODESET27 129				CPX_PARAM_NETEPOPT 96
CPX_PARAM_NODESET28 130				CPX_PARAM_NETEPRHS 96
CPX_PARAM_NODESET29 131				CPX_PARAM_NETFIND 97
CPX_PARAM_NODESET30 132				CPX_PARAM_NETITLIM 98
CPX_PARAM_NODESET31 133				CPX_PARAM_NETPPRIIND 98
CPXPARAM_MIP_Limits_RampupTimeLimit 128	CPXPARAM_MIP_Limits_Solutions 79	CPXPARAM_OptimalityTarget 106	CPXPARAM_TimeLimit 159	
CPXPARAM_MIP_Limits_StrongCand 154	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_SOLUTIONTYPE 152	CPXPARAM_Tune_DetTimeLimit 160	
CPXPARAM_MIP_Limits_TreeMemory 160	CPXPARAM_MIP_OrderType 91	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_Display 162	
CPXPARAM_MIP_Pool_AbsGap 146	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Measure 163	
CPXPARAM_MIP_Pool_Intensity 149		CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Repeat 164	
		CPX_PARAM_SUBALG 99	CPXPARAM_Tune_TimeLimit 165	
		CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_WorkDir 167	
		CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkMem 168	
		CPX_PARAM_THREADS 157	CraInd 50	
		CPX_PARAM_TILIM 159		

ML to speed up MILP solvers

- Modern MILP solvers expose **hundreds** of parameters
- Defaults are rarely optimal: instance families vary widely in structure

ML to speed up MILP solvers

- Modern MILP solvers expose **hundreds** of parameters
- Defaults are rarely optimal: instance families vary widely in structure
- Small changes can yield **large runtime** differences

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETTILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 160	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 20	CPX_PARAM_FPHEUR 72
CPX_PARAM_NODELIM 101				FRACCAND 73
CPX_PARAM_NODESEL 102				FRACCUTS 73
CPX_PARAM_NODELIM 101				FRACPASS 74
CPX_PARAM_NODESEL 102				GUBCOVERS 75
CPX_PARAM_NODELIM 101				HEURFREQ 76
CPX_PARAM_NODESEL 102				IMPLBD 76
CPX_PARAM_NODELIM 101				INTSOLFILEPREFIX 78
CPX_PARAM_NODESEL 102				INTSOLLIM 79
CPX_PARAM_NODELIM 101				ITLIM 80
CPX_PARAM_NODESEL 102				LANDPCUTS 82
CPX_PARAM_NODELIM 101				LBHEUR 81
CPX_PARAM_NODESEL 102				LPMETHOD 136
CPX_PARAM_NODELIM 101				MCFCUTS 82
CPX_PARAM_NODESEL 102				MEMORYEMPHASIS 83
CPX_PARAM_NODELIM 101				MIPCBREDLP 84
CPX_PARAM_NODESEL 102				MIPDISPLAY 85
CPX_PARAM_NODELIM 101				MIPEMPHASIS 87
CPX_PARAM_NODESEL 102				MIPINTERVAL 88
CPX_PARAM_NODELIM 101				MIPKAPPASTATS 89
CPX_PARAM_NODESEL 102				MIPORDIND 90
CPX_PARAM_NODELIM 101				MIPORDTYPE 91
CPX_PARAM_NODESEL 102				CPX_PARAM_MIPSEARCH 92
CPX_PARAM_NODELIM 101				CPX_PARAM_MIQCPSTRAT 93
CPX_PARAM_NODESEL 102				CPX_PARAM_MIRCUTS 94
CPX_PARAM_NODELIM 101				CPX_PARAM_MPSSLONGNUM 94
CPX_PARAM_NODESEL 102				CPX_PARAM_NETDISPLAY 95
CPX_PARAM_NODELIM 101				CPX_PARAM_NETEPOPT 96
CPX_PARAM_NODESEL 102				CPX_PARAM_NETEPRHS 96
CPX_PARAM_NODELIM 101				CPX_PARAM_NETFIND 97
CPX_PARAM_NODESEL 102				CPX_PARAM_NETITLIM 98
CPX_PARAM_NODELIM 101				CPX_PARAM_NETPPRIIND 98
CPX_PARAM_PREPASS 121	CPXPARAM_MIP_Limits_RampupTimeLimit 128	CPXPARAM_OptimalityTarget 106	CPXPARAM_TimeLimit 159	
CPX_PARAM_PRESLVND 122	CPXPARAM_MIP_Limits_Solutions 79	CPX_PARAM_SOLUTIONTYPE 152	CPXPARAM_Tune_DetTimeLimit 160	
CPX_PARAM_PRICELIM 123	CPXPARAM_MIP_Limits_StrongCand 154	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_Display 162	
CPX_PARAM_PROBE 123	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Measure 163	
CPX_PARAM_PROBEDETTIME 124	CPXPARAM_MIP_Limits_TreeMemory 160	CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Repeat 164	
CPX_PARAM_PROBETIME 124	CPXPARAM_MIP_OrderType 91	CPX_PARAM_SUBALG 99	CPXPARAM_Tune_TimeLimit 165	
CPX_PARAM_QPMAKEPSDIND 125	CPXPARAM_MIP_Pool_AbsGap 146	CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_WorkDir 167	
CPX_PARAM_QPMETHOD 138	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkMem 168	
CPX_PARAM_QPNZREADLIM 126	CPXPARAM_MIP_Pool_Intensity 149	CPX_PARAM_THREADS 157	CraInd 50	
		CPX_PARAM_TILIM 159		

ML to speed up MILP solvers

- Modern MILP solvers expose **hundreds** of parameters
- Defaults are rarely optimal: instance families vary widely in structure
- Small changes can yield **large runtime** differences
- **Many** ways to integrate ML into solvers

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETTILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 160	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 20	CPX_PARAM_FPHEUR 72
CPX_PARAM_NODESEL 102				FRACCAND 73
CPX_PARAM_NODESEL 102				FRACCUTS 73
CPX_PARAM_NODESEL 102				FRACPASS 74
CPX_PARAM_NODESEL 102				GUBCOVERS 75
CPX_PARAM_NODESEL 102				HEURFREQ 76
CPX_PARAM_NODESEL 102				IMPLBD 76
CPX_PARAM_NODESEL 102				INTSOLFILEPREFIX 78
CPX_PARAM_NODESEL 102				INTSOLLIM 79
CPX_PARAM_NODESEL 102				ITLIM 80
CPX_PARAM_NODESEL 102				LANDPCUTS 82
CPX_PARAM_NODESEL 102				LBHEUR 81
CPX_PARAM_NODESEL 102				LPMETHOD 136
CPX_PARAM_NODESEL 102				MCFCUTS 82
CPX_PARAM_NODESEL 102				MEMORYEMPHASIS 83
CPX_PARAM_NODESEL 102				MIPCBREDLP 84
CPX_PARAM_NODESEL 102				MIPDISPLAY 85
CPX_PARAM_NODESEL 102				MIPEMPHASIS 87
CPX_PARAM_NODESEL 102				MIPINTERVAL 88
CPX_PARAM_NODESEL 102				MIPKAPPASTATS 89
CPX_PARAM_NODESEL 102				MIPORDIND 90
CPX_PARAM_NODESEL 102				MIPORDTYPE 91
CPX_PARAM_PREPASS 121	CPXPARAM_MIP_Limits_RampupTimeLimit 128	CPXPARAM_OptimalityTarget 106	CPXPARAM_TimeLimit 159	CPX_PARAM_MIPSEARCH 92
CPX_PARAM_PRESLVND 122	CPXPARAM_MIP_Limits_Solutions 79	CPX_PARAM_SOLUTIONTYPE 152	CPXPARAM_Tune_DetTimeLimit 160	CPX_PARAM_MIQCPSTRAT 93
CPX_PARAM_PRICELIM 123	CPXPARAM_MIP_Limits_StrongCand 154	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_Display 162	CPX_PARAM_MIRCUTS 94
CPX_PARAM_PROBE 123	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Measure 163	CPX_PARAM_MPSSLONGNUM 94
CPX_PARAM_PROBEDETTIME 124	CPXPARAM_MIP_Limits_TreeMemory 160	CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Repeat 164	CPX_PARAM_NETDISPLAY 95
CPX_PARAM_PROBETIME 124	CPXPARAM_MIP_OrderType 91	CPX_PARAM_SUBALG 99	CPXPARAM_Tune_TimeLimit 165	CPX_PARAM_NETEPOPT 96
CPX_PARAM_QPMAKEPSDIND 125	CPXPARAM_MIP_Pool_AbsGap 146	CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_WorkDir 167	CPX_PARAM_NETEPRHS 96
CPX_PARAM_QPMETHOD 138	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkMem 168	CPX_PARAM_NETFIND 97
CPX_PARAM_QPNZREADLIM 126	CPXPARAM_MIP_Pool_Intensity 149	CPX_PARAM_THREADS 157	CraInd 50	CPX_PARAM_NETITLIM 98
		CPX_PARAM_TILIM 159		CPX_PARAM_NETPPRIIND 98

ML to speed up MILP solvers

- Modern MILP solvers expose **hundreds** of parameters
- Defaults are rarely optimal: instance families vary widely in structure
- Small changes can yield **large runtime** differences
- **Many** ways to integrate ML into solvers
 - Potential for ML to provide **significant** speedups over defaults

CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETTILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 160	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 20	CPX_PARAM_FPHEUR 72
CPX_PARAM_NODESEL 102				FRACCAND 73
CPX_PARAM_NODESEL 102				FRACCUTS 73
CPX_PARAM_NODESEL 102				FRACPASS 74
CPX_PARAM_NODESEL 102				GUBCOVERS 75
CPX_PARAM_NODESEL 102				HEURFREQ 76
CPX_PARAM_NODESEL 102				IMPLBD 76
CPX_PARAM_NODESEL 102				INTSOLFILEPREFIX 78
CPX_PARAM_NODESEL 102				INTSOLLIM 79
CPX_PARAM_NODESEL 102				ITLIM 80
CPX_PARAM_NODESEL 102				LANDPCUTS 82
CPX_PARAM_NODESEL 102				LBHEUR 81
CPX_PARAM_NODESEL 102				LPMETHOD 136
CPX_PARAM_NODESEL 102				MCFCUTS 82
CPX_PARAM_NODESEL 102				MEMORYEMPHASIS 83
CPX_PARAM_NODESEL 102				MIPCBREDLP 84
CPX_PARAM_NODESEL 102				MIPDISPLAY 85
CPX_PARAM_NODESEL 102				MIPEMPHASIS 87
CPX_PARAM_NODESEL 102				MIPINTERVAL 88
CPX_PARAM_NODESEL 102				MIPKAPPASTATS 89
CPX_PARAM_NODESEL 102				MIPORDIND 90
CPX_PARAM_NODESEL 102				MIPORDTYPE 91
CPX_PARAM_PREPASS 121	CPXPARAM_MIP_Limits_RampupTimeLimit 128	CPXPARAM_OptimalityTarget 106	CPXPARAM_TimeLimit 159	CPX_PARAM_MIPSEARCH 92
CPX_PARAM_PRESLVND 122	CPXPARAM_MIP_Limits_Solutions 79	CPX_PARAM_SOLUTIONTYPE 152	CPXPARAM_Tune_DetTimeLimit 160	CPX_PARAM_MIQCPSTRAT 93
CPX_PARAM_PRICELIM 123	CPXPARAM_MIP_Limits_StrongCand 154	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_Display 162	CPX_PARAM_MIRCUTS 94
CPX_PARAM_PROBE 123	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Measure 163	CPX_PARAM_MPSSLONGNUM 94
CPX_PARAM_PROBEDETTIME 124	CPXPARAM_MIP_Limits_TreeMemory 160	CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Repeat 164	CPX_PARAM_NETDISPLAY 95
CPX_PARAM_PROBETIME 124	CPXPARAM_MIP_OrderType 91	CPX_PARAM_SUBALG 99	CPXPARAM_Tune_TimeLimit 165	CPX_PARAM_NETEPOPT 96
CPX_PARAM_QPMAKEPSDIND 125	CPXPARAM_MIP_Pool_AbsGap 146	CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_WorkDir 167	CPX_PARAM_NETEPRHS 96
CPX_PARAM_QPMETHOD 138	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkMem 168	CPX_PARAM_NETFIND 97
CPX_PARAM_QPNZREADLIM 126	CPXPARAM_MIP_Pool_Intensity 149	CPX_PARAM_THREADS 157	CraInd 50	CPX_PARAM_NETITLIM 98
		CPX_PARAM_TILIM 159		CPX_PARAM_NETPPRIIND 98

Outline

1. Background
 - i. **Branch-and-cut (B&C) algorithm**
 - ii. Brief overview of ML for B&C: non-GenAI
2. LLMs for speeding up solvers
3. Takeaways

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

- Explore **search tree** of restricted MILP subproblems



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
 - If integer-feasible: incumbent solution

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
 - If integer-feasible: incumbent solution
- **Branch** by choosing a variable and splitting its domain

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
 - If integer-feasible: incumbent solution
- **Branch** by choosing a variable and splitting its domain

**Variable
selection policy**

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
 - If integer-feasible: incumbent solution
- **Branch** by choosing a variable and splitting its domain ←

**Variable
selection policy**
- **Prune** nodes whose bound can't beat incumbent

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
 - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
 - If integer-feasible: incumbent solution
- **Branch** by choosing a variable and splitting its domain ←

**Variable
selection policy**
- **Prune** nodes whose bound can't beat incumbent
- Terminate when all nodes **pruned** or **proven optimal**

Background: Branch-and-cut algorithm

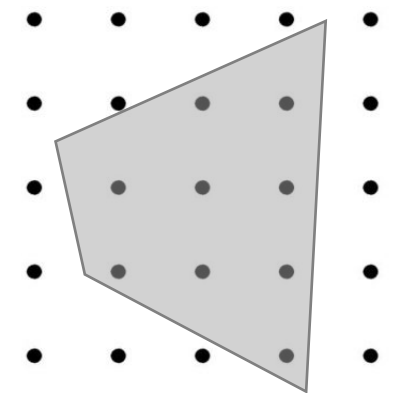
Uses guidance from LP relaxations to guide search

MILP

maximize $\mathbf{c}^T \mathbf{x}$

subject to $A\mathbf{x} \leq \mathbf{b}$

$x_i \in \mathbb{Z}$ for all $i \in I$



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

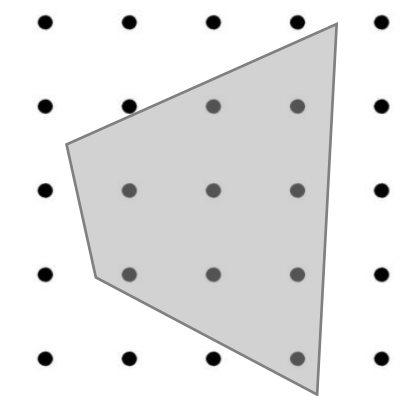
$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $c^T x$

subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

$x_i \in \mathbb{Z}$ for all $i \in I$

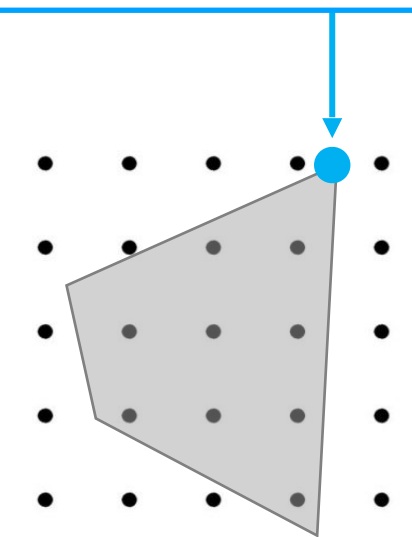
Linear programming (LP) relaxation

maximize $c^T x$

subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

LP optimal solution



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

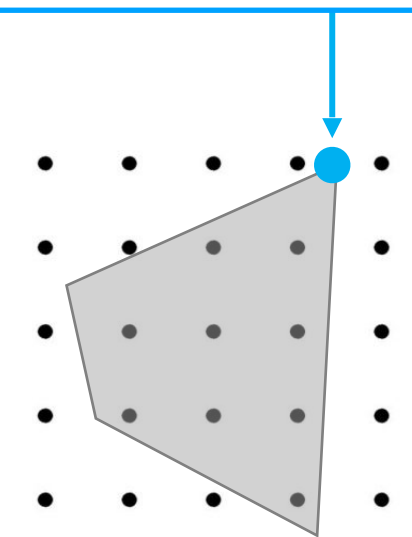
maximize $c^T x$

subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

LP optimal solution



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $c^T x$

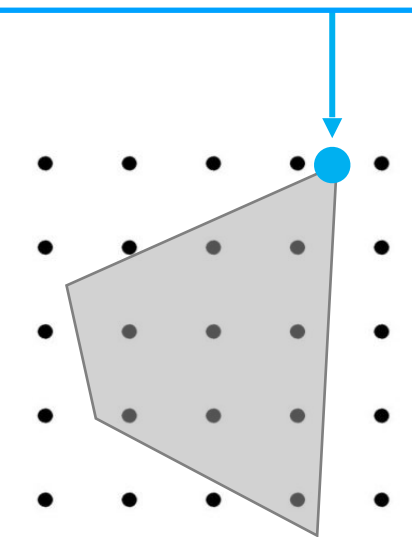
subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

- Separate LP optimal solution

LP optimal solution



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $c^T x$

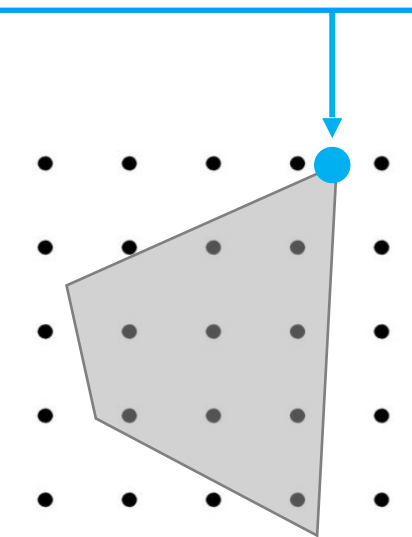
subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

- Separate LP optimal solution
- Don't separate any integer point

LP optimal solution



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $c^T x$

subject to $Ax \leq b$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $c^T x$

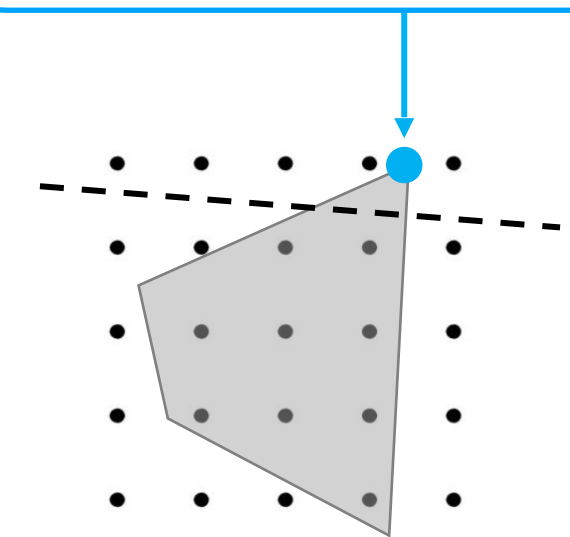
subject to $Ax \leq b$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

- Separate LP optimal solution
- Don't separate any integer point

LP optimal solution



Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $\mathbf{c}^T \mathbf{x}$

subject to $A\mathbf{x} \leq \mathbf{b}$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $\mathbf{c}^T \mathbf{x}$

subject to $A\mathbf{x} \leq \mathbf{b}$

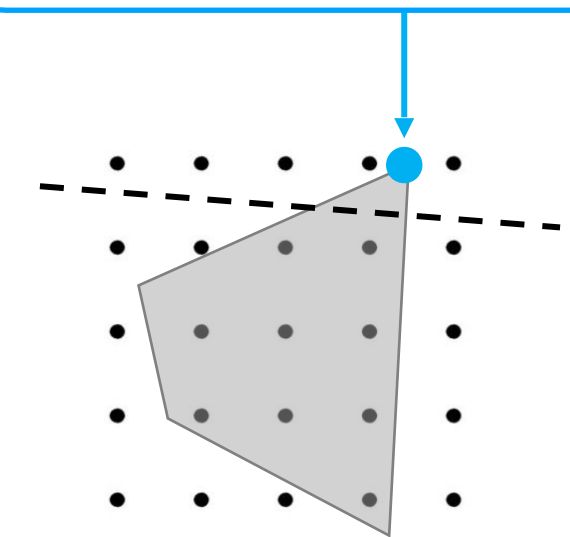
~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

- Separate LP optimal solution
- Don't separate any integer point

Many different families of CPs; which to use when?

LP optimal solution



Outline

1. Background
 - i. Branch-and-cut (B&C) algorithm
 - ii. Brief overview of ML for B&C: non-GenAI**
2. LLMs for speeding up solvers
3. Takeaways

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi
 - θ are parameters of a neural network embedded in solver

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi
 - θ are parameters of a neural network embedded in solver
- Specify distribution D over MILPs z (models **application domain**)

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi
 - θ are parameters of a neural network embedded in solver
- Specify distribution D over MILPs z (models **application domain**)
- Choose a **performance metric** $c(z, \theta)$; e.g., runtime

General setup

ML for MILP solvers

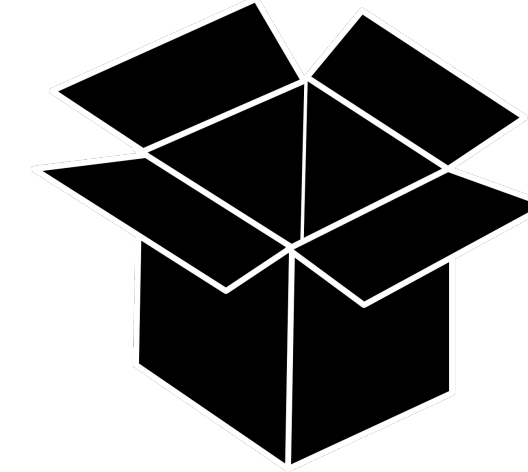
- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi
 - θ are parameters of a neural network embedded in solver
- Specify distribution D over MILPs z (models **application domain**)
- Choose a **performance metric** $c(z, \theta)$; e.g., runtime
- **Ultimate goal:** minimize $\mathbb{E}_{z \sim D} [c(z, \theta)]$ (proxy of *future* cost on unseen MILPs)

General setup

ML for MILP solvers

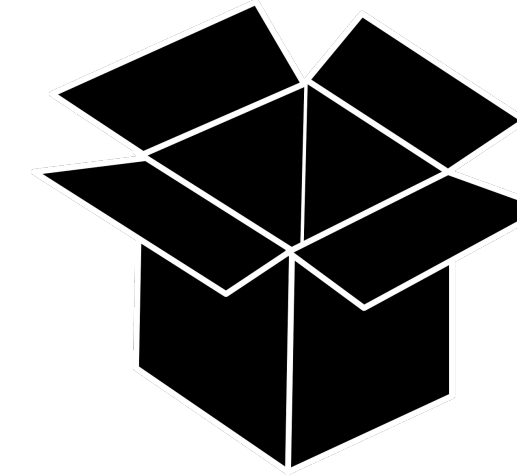
- Define a parameterized solver $A(\theta)$. E.g.:
 - θ are parameters exposed by Gurobi
 - θ are parameters of a neural network embedded in solver
- Specify distribution D over MILPs z (models **application domain**)
- Choose a **performance metric** $c(z, \theta)$; e.g., runtime
- **Ultimate goal:** minimize $\mathbb{E}_{z \sim D} [c(z, \theta)]$ (proxy of *future* cost on unseen MILPs)
 - Can learn *offline* θ or *instance-aware* $\theta(z)$ configuration

Blackbox algorithm configuration



- Early work: treat solver (largely) as a black-box; learn from evaluations

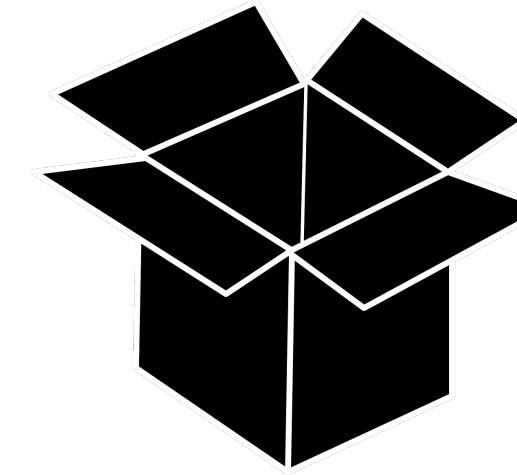
Blackbox algorithm configuration



- Early work: treat solver (largely) as a black-box; learn from evaluations
- Small subset of examples:
 - **ParamILS**: iterated local search over parameter settings

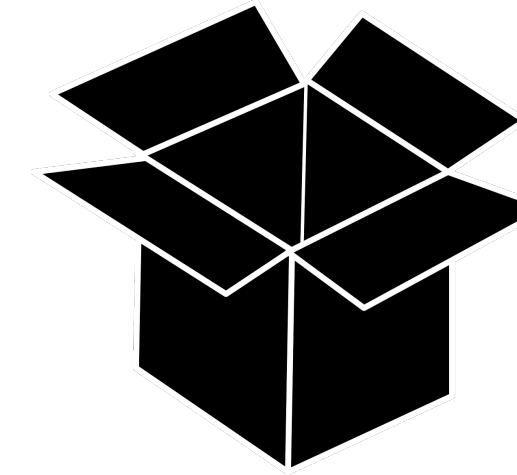
Hutter, Hoos, Leyton-Brown, Stützle, JAIR'09

Blackbox algorithm configuration



- Early work: treat solver (largely) as a black-box; learn from evaluations
- Small subset of examples:
 - **ParamILS**: iterated local search over parameter settings
Hutter, Hoos, Leyton-Brown, Stützle, JAIR'09
 - **SMAC**: model-based search with surrogate predictions
Hutter, Hoos, Leyton-Brown, LION'11

Blackbox algorithm configuration



- Early work: treat solver (largely) as a black-box; learn from evaluations
- Small subset of examples:
 - **ParamILS**: iterated local search over parameter settings
Hutter, Hoos, Leyton-Brown, Stützle, JAIR'09
 - **SMAC**: model-based search with surrogate predictions
Hutter, Hoos, Leyton-Brown, LION'11
 - **Portfolio-based algorithm selection**
Lobjois, Lemaître, AAAI'98; Gomes, Selman, AI'01; Xu, Hoos, Leyton-Brown, AAAI'10; Kadioglu et al., ECAI'10, Sandholm, Handbook of Market Design'13

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations
- **Hydra** iteratively grows portfolio via targeted reconfiguration

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations
- **Hydra** iteratively grows portfolio via targeted reconfiguration
 - Tune new member against instances current portfolio **solves poorly**

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations
- **Hydra** iteratively grows portfolio via targeted reconfiguration
 - Tune new member against instances current portfolio **solves poorly**
- Instance features enable **per-instance** selection (static & quick probing runs)

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration



- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations
- **Hydra** iteratively grows portfolio via targeted reconfiguration
 - Tune new member against instances current portfolio **solves poorly**
- Instance features enable **per-instance** selection (static & quick probing runs)
 - Choose configuration before solving each instance

Configuration of solver components

Next gen: don't treat solver as blackbox; adapt to solver components. E.g.:

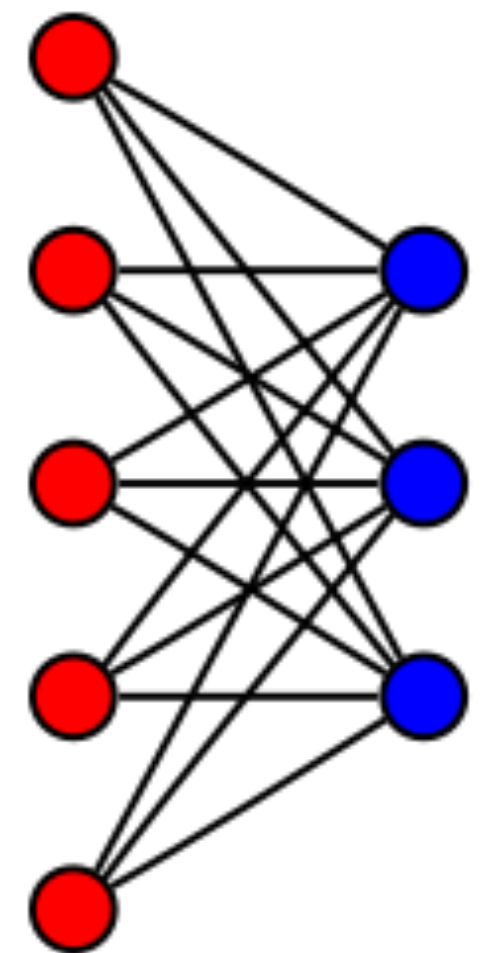
Cut selection	Variable selection	Node selection
Tang et al., ICML'20 Balcan et al., NeurIPS'21 Balcan et al., NeurIPS'22 Paulus et al., ICML'22 Wang et al., ICLR'23 Li et al., NeurIPS'23 Deza, Khalil, IJCAI'23 Ling et al., AAI'24 Cheng, Basu, NeurIPS'24 Cheng et al., NeurIPS'24 ...	Khalil et al., AAI'16 Alvarez et al., INFORMS JoC'17 Balcan et al., ICML'18 Gasse et al., NeurIPS'19 Gupta et al., NeurIPS'20 Zarpellon et al., AAI'21 Scavuzzo et al., NeurIPS'22 ...	He et al., NeurIPS'14 Labassi et al., NeurIPS'22 Zhang et al., ICLR'25 ... 

Configuration of solver components

Next gen: don't treat solver as blackbox; adapt to solver components. E.g.:

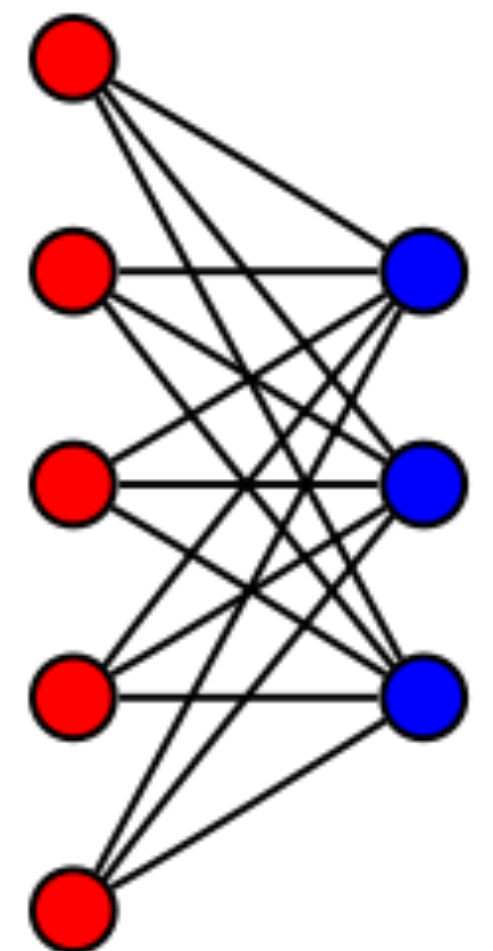
Cut selection	Variable selection	Node selection
Tang et al., ICML'20 Balcan et al., NeurIPS'21 Balcan et al., NeurIPS'22 Paulus et al., ICML'22 Wang et al., ICLR'23 Li et al., NeurIPS'23 Deza, Khalil, IJCAI'23 Ling et al., AAI'24 Cheng, Basu, NeurIPS'24 Cheng et al., NeurIPS'24 ...	Khalil et al., AAI'16 Alvarez et al., INFORMS JoC'17 Balcan et al., ICML'18 Gasse et al., NeurIPS'19 Gupta et al., NeurIPS'20 Zarpellon et al., AAI'21 Scavuzzo et al., NeurIPS'22 ...	He et al., NeurIPS'14 Labassi et al., NeurIPS'22 Zhang et al., ICLR'25 ... 

Graph neural networks for variable selection



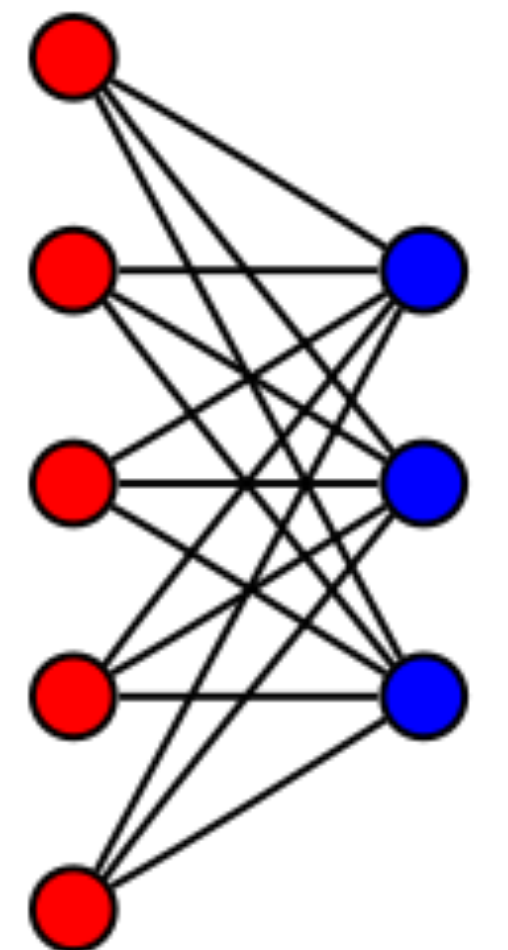
Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**



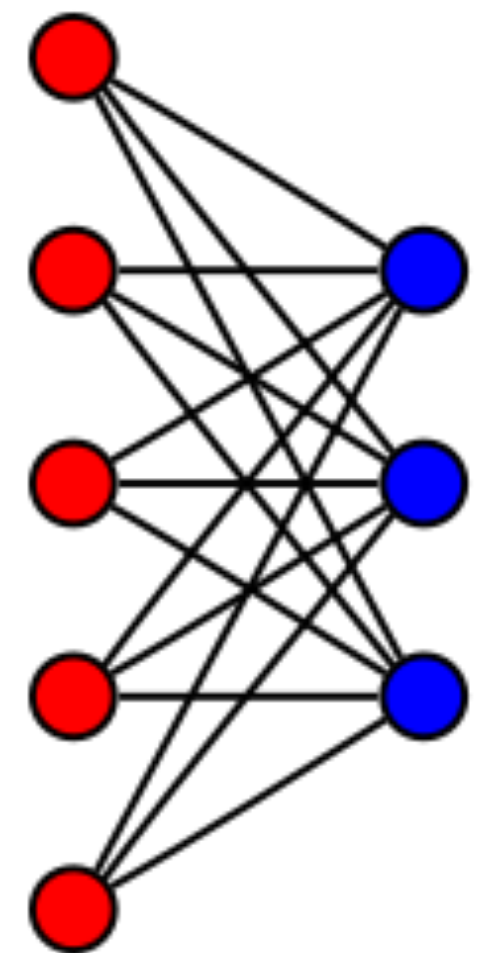
Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
 - Use bipartite **graph neural network** as a variable selection policy



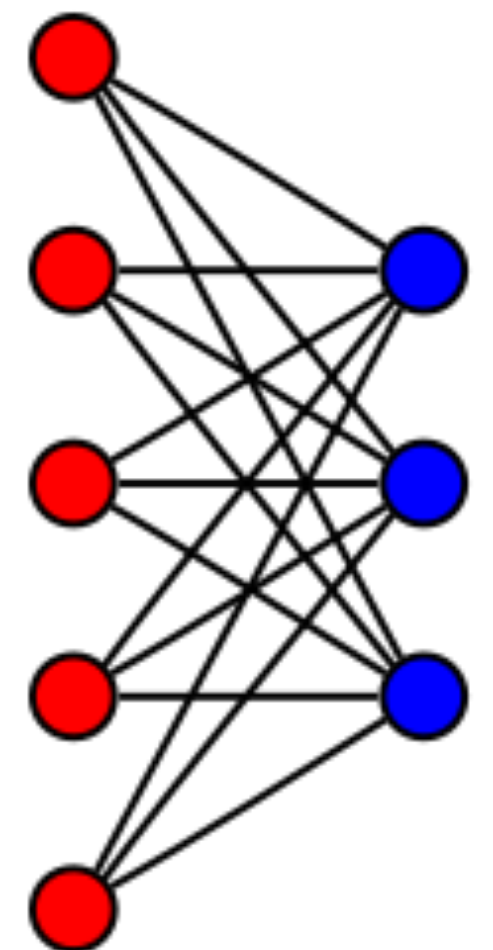
Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
 - Use bipartite **graph neural network** as a variable selection policy
- Training: behavioral cloning of strong branching (expensive **gold standard**)



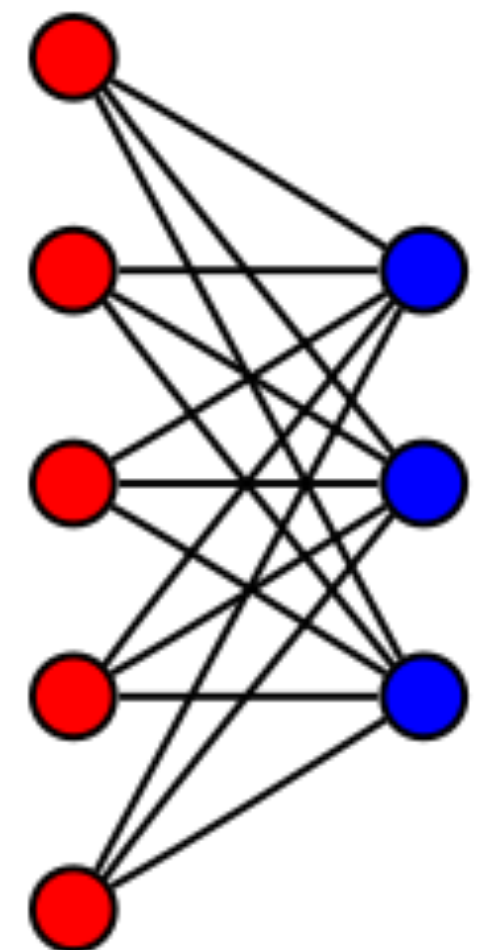
Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
 - Use bipartite **graph neural network** as a variable selection policy
- Training: behavioral cloning of strong branching (expensive **gold standard**)
- Integrated in SCIP; four NP-hard benchmarks



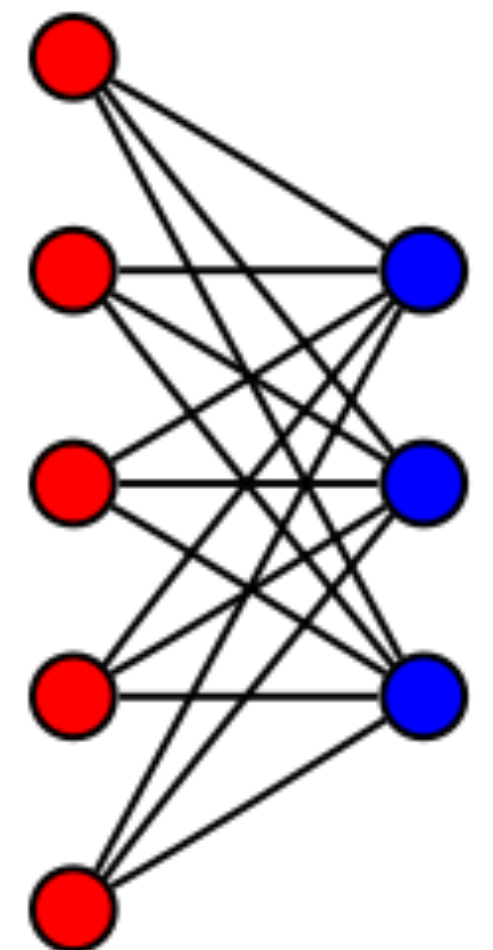
Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
 - Use bipartite **graph neural network** as a variable selection policy
- Training: behavioral cloning of strong branching (expensive **gold standard**)
- Integrated in SCIP; four NP-hard benchmarks
- Results:
 - Best imitation accuracy among ML baselines



Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
 - Use bipartite **graph neural network** as a variable selection policy
- Training: behavioral cloning of strong branching (expensive **gold standard**)
- Integrated in SCIP; four NP-hard benchmarks
- Results:
 - Best imitation accuracy among ML baselines
 - Generally faster than SCIP default; good size generalization



Benchmarks

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
 - More than 35 distributions across 13 domains

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
 - More than 35 distributions across 13 domains
 - Includes synthetic and real-world domains, multiple hardness levels

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
 - More than 35 distributions across 13 domains
 - Includes synthetic and real-world domains, multiple hardness levels
- **MILP-Evolve** [Li et al. ICLR’25]

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
 - More than 35 distributions across 13 domains
 - Includes synthetic and real-world domains, multiple hardness levels
- **MILP-Evolve** [Li et al. ICLR’25]
 - “Evolving” pipeline for generating new MILP families

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
 - Benchmark set: 240 instances, solvable, numerically stable
 - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
 - More than 35 distributions across 13 domains
 - Includes synthetic and real-world domains, multiple hardness levels
- **MILP-Evolve** [Li et al. ICLR’25]
 - “Evolving” pipeline for generating new MILP families
 - Designed to be highly diverse to mimic real-world optimization scenarios

Outline

1. Background
2. LLMs for speeding up solvers
 - i. **Solver configuration**
 - ii. Evolutionary search for MILP heuristics
3. Takeaways

Key challenge

Conventional data-driven approaches require a lot of compute

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

1. Gather a **training set** of MILPs

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

1. Gather a **training set** of MILPs
2. Find **configuration(s) with good average runtime** over training set
3. Hope for a good runtime from the same application (or distribution)

Key challenge: Evaluating one configuration's average runtime

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

1. Gather a **training set** of MILPs
2. Find **configuration(s) with good average runtime** over training set
3. Hope for a good runtime from the same application (or distribution)

Key challenge: Evaluating one configuration's average runtime requires **solving** every MILP in the training set using that configuration

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

1. Gather a **training set** of MILPs
2. Find **configuration(s) with good average runtime** over training set
3. Hope for a good runtime from the same application (or distribution)

Key challenge: Evaluating one configuration's average runtime requires **solving** every MILP in the training set using that configuration

Key question: Can we generate problem-specific cutting plane configurations

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

1. Gather a **training set** of MILPs
2. Find **configuration(s) with good average runtime** over training set
3. Hope for a good runtime from the same application (or distribution)

Key challenge: Evaluating one configuration's average runtime requires **solving** every MILP in the training set using that configuration

Key question: Can we generate problem-specific cutting plane configurations with **little to no historical data and compute?**

Our contributions

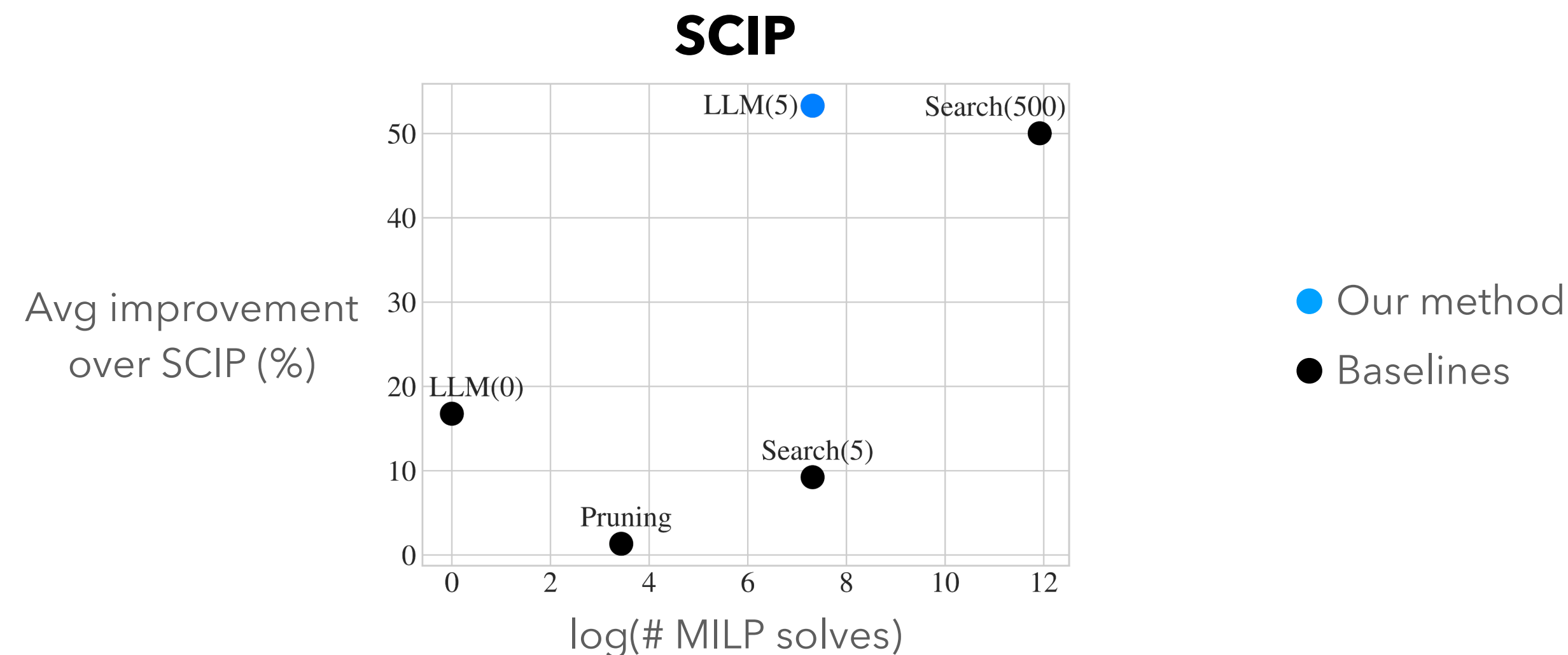
- **First LLM-based framework** to configure MILP solvers

Our contributions

- **First LLM-based framework** to configure MILP solvers
- **Consistent improvement** over solver default (SCIP and Gurobi)

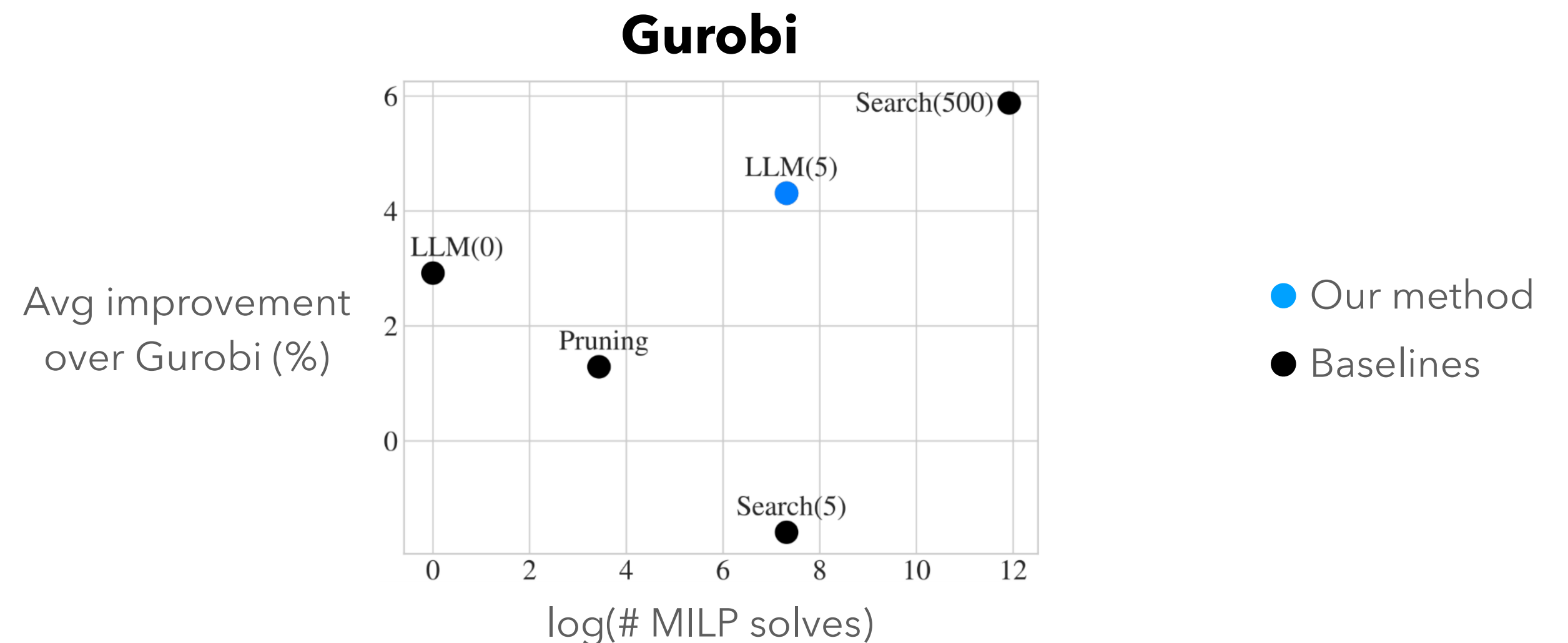
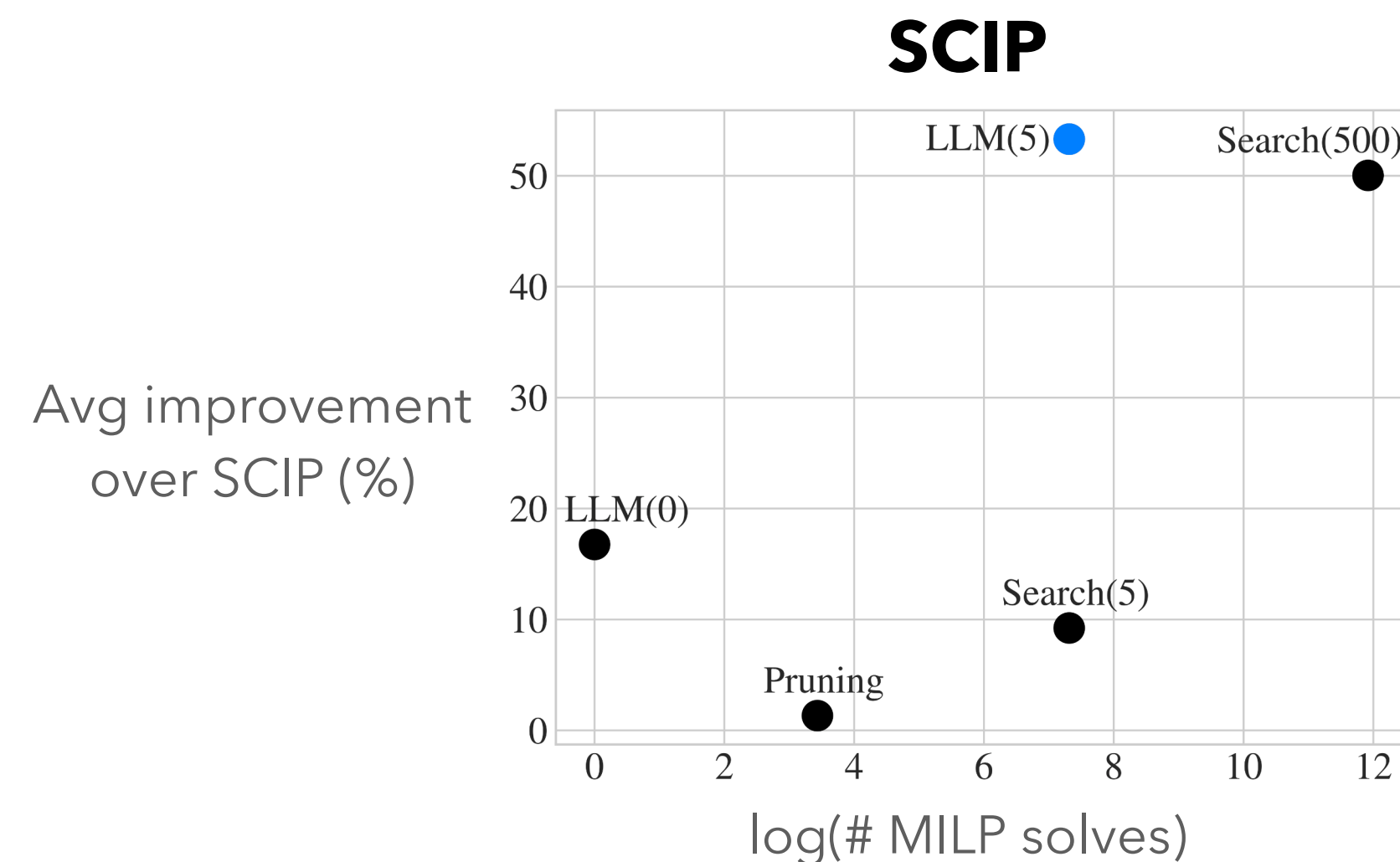
Our contributions

- **First LLM-based framework** to configure MILP solvers
- **Consistent improvement** over solver default (SCIP and Gurobi)
 - Pareto-optimal compared to baseline methods



Our contributions

- **First LLM-based framework** to configure MILP solvers
- **Consistent improvement** over solver default (SCIP and Gurobi)
 - Pareto-optimal compared to baseline methods



Why LLMs

- LLMs are powerful, but they can't do everything

Why LLMs

- LLMs are powerful, but they can't do everything
- They are good at **information retrieval**

Why LLMs

- LLMs are powerful, but they can't do everything
- They are good at **information retrieval**
- There's a **rich literature** on cutting planes



Challenges to using LLMs

- LLM output can be highly **unstable**

Challenges to using LLMs

- LLM output can be highly **unstable**
- Cutting plane separators are **solver-specific**

Challenges to using LLMs

- LLM output can be highly **unstable**
- Cutting plane separators are **solver-specific**
 - Details of solver separators are not always available

> CoverCuts

Cover cut generation

- Type: `int`
- Default value: `-1`
- Minimum value: `-1`
- Maximum value: `2`

Controls cover cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the [Cuts](#) parameter.

LLM for cold-start configuration pipeline

LLM for cold-start configuration pipeline

Problem description

Text summary

LaTeX model

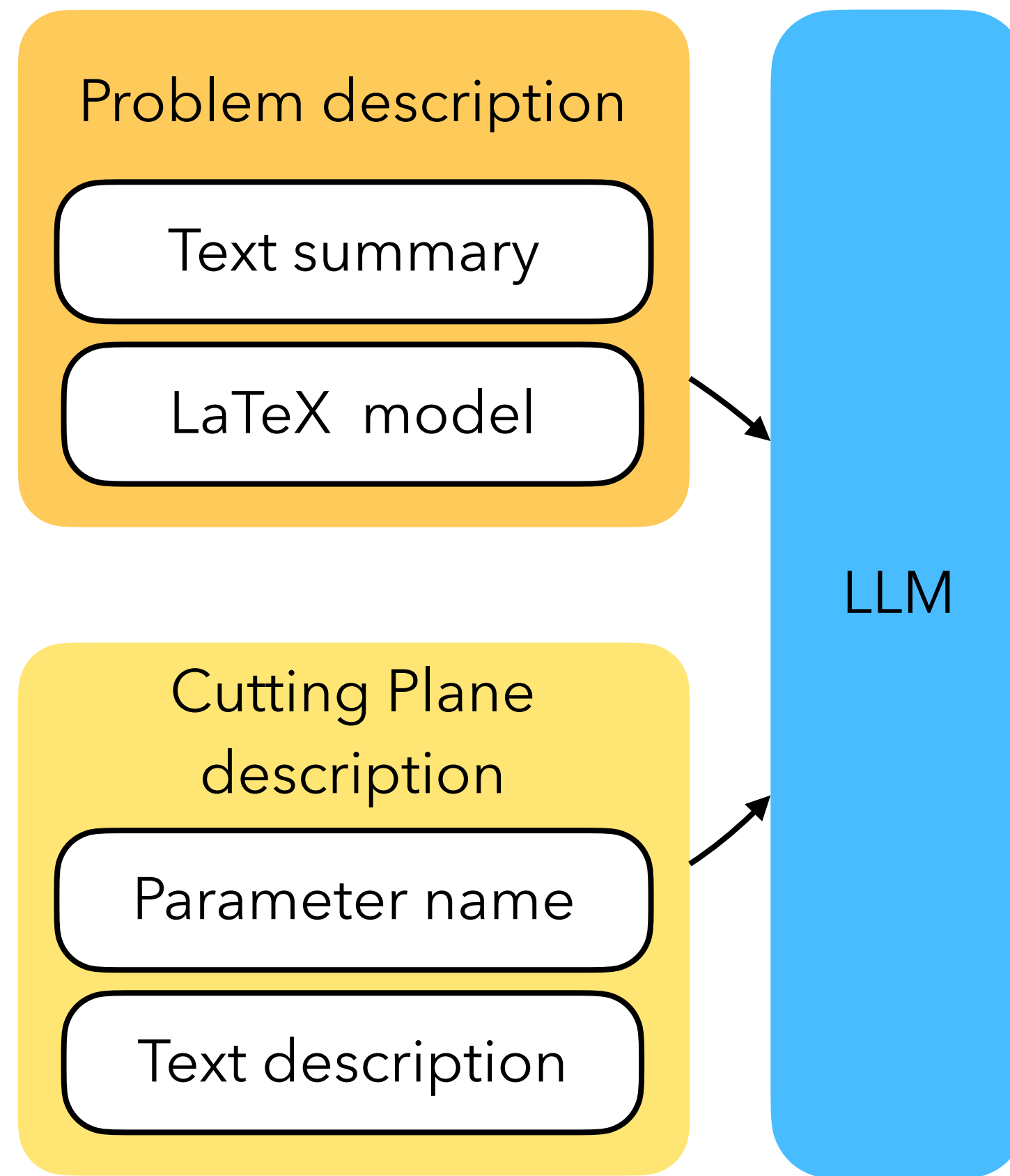
Cutting Plane
description

Parameter name

Text description

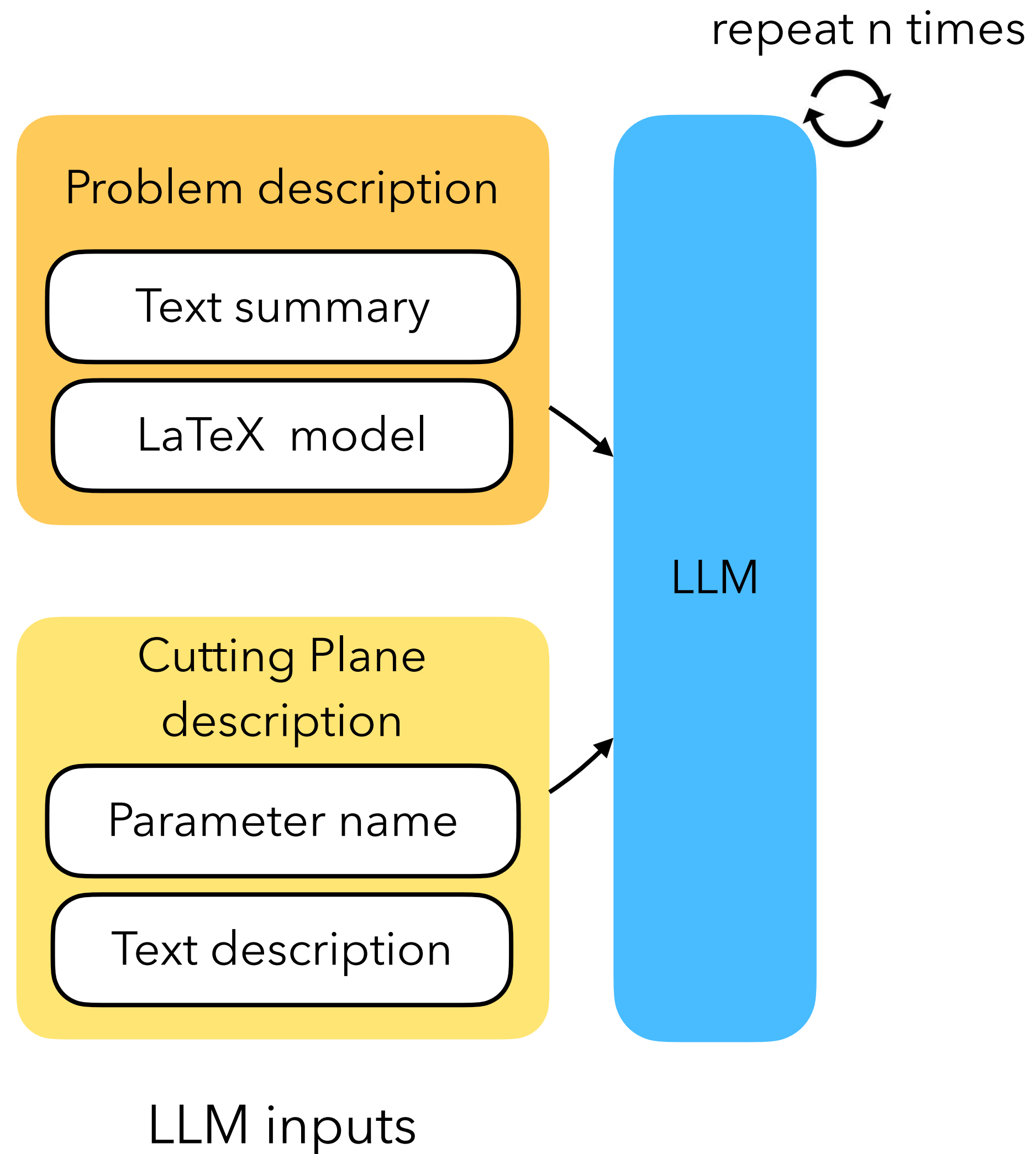
LLM inputs

LLM for cold-start configuration pipeline

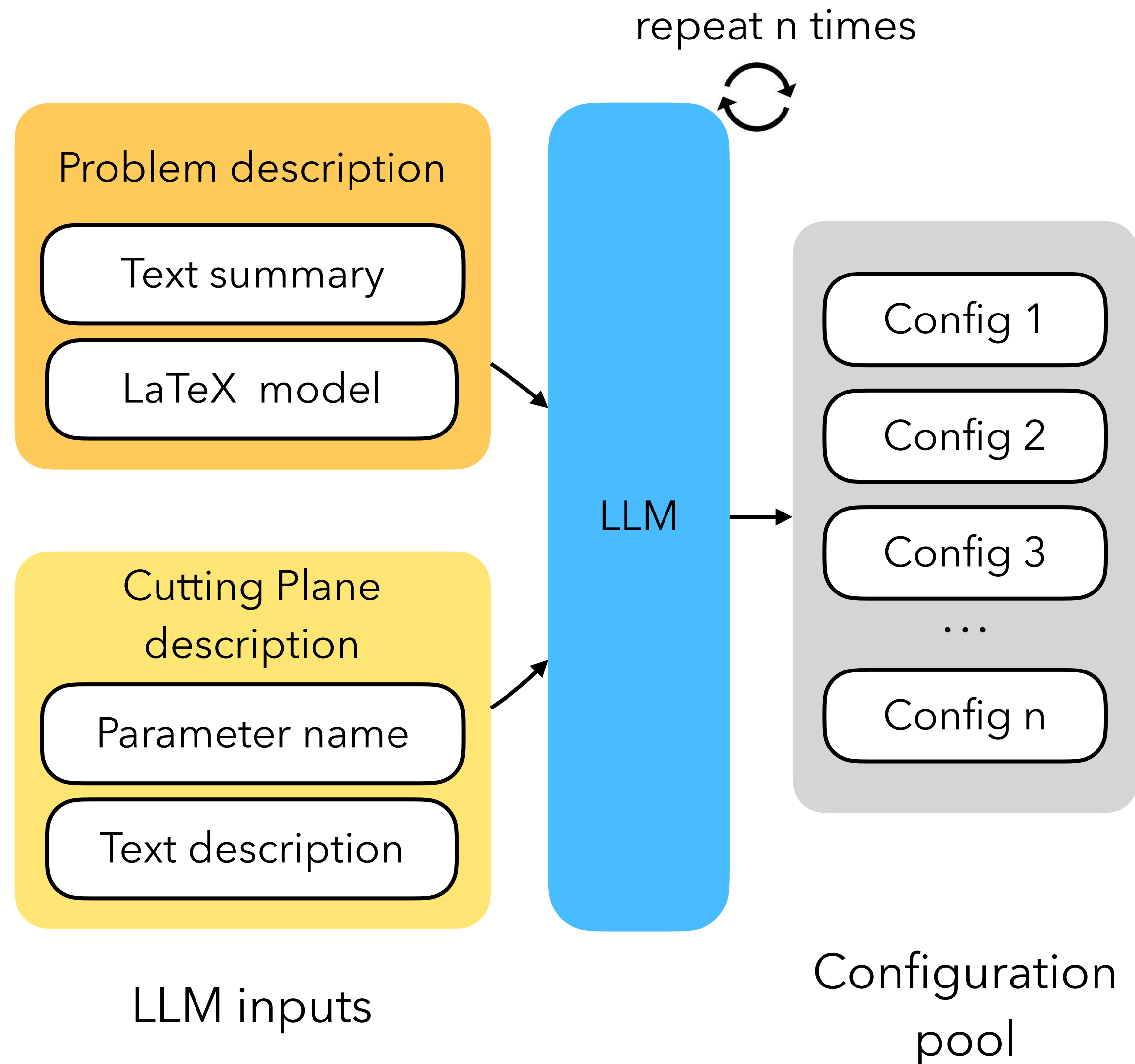


LLM inputs

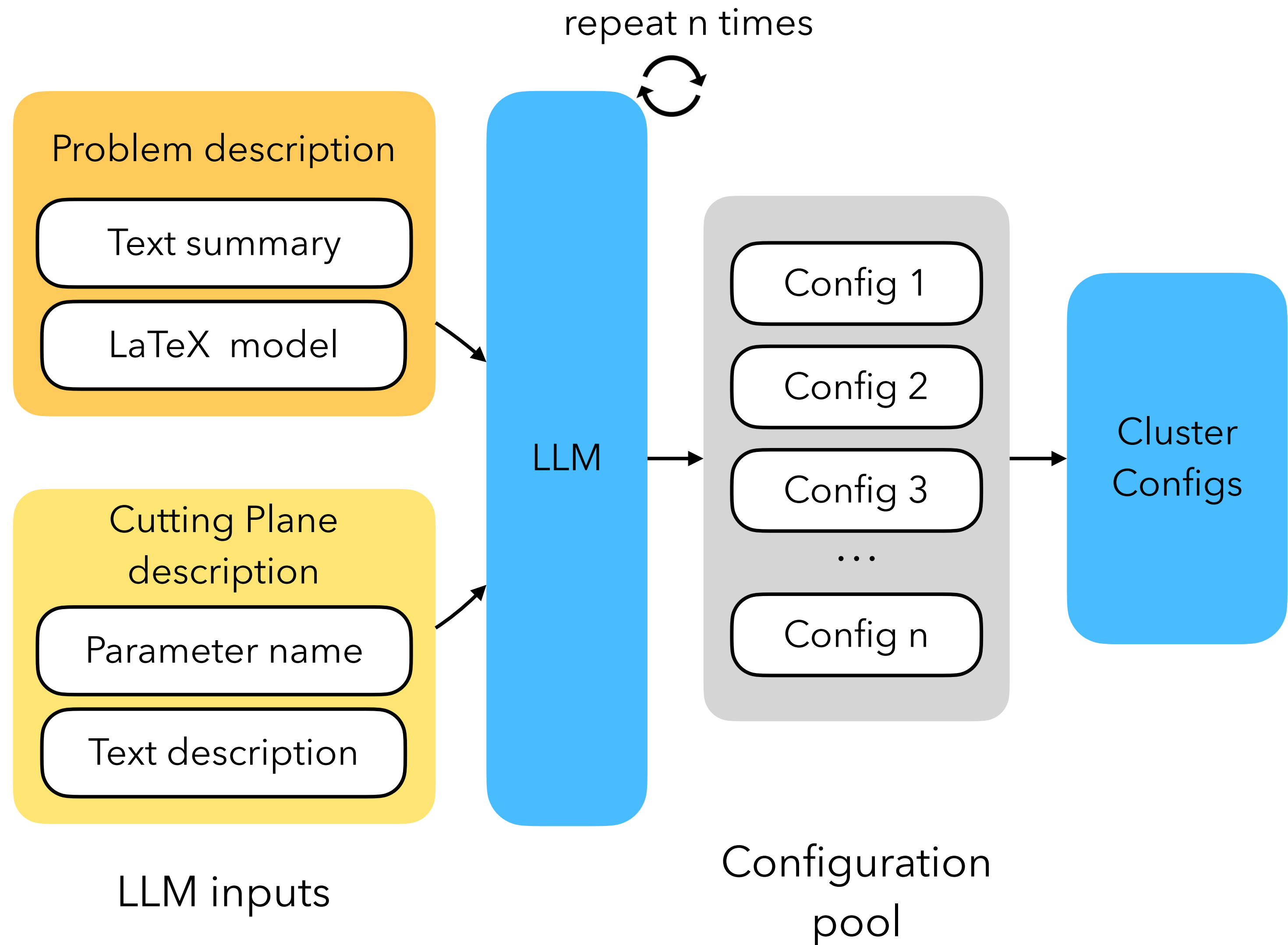
LLM for cold-start configuration pipeline



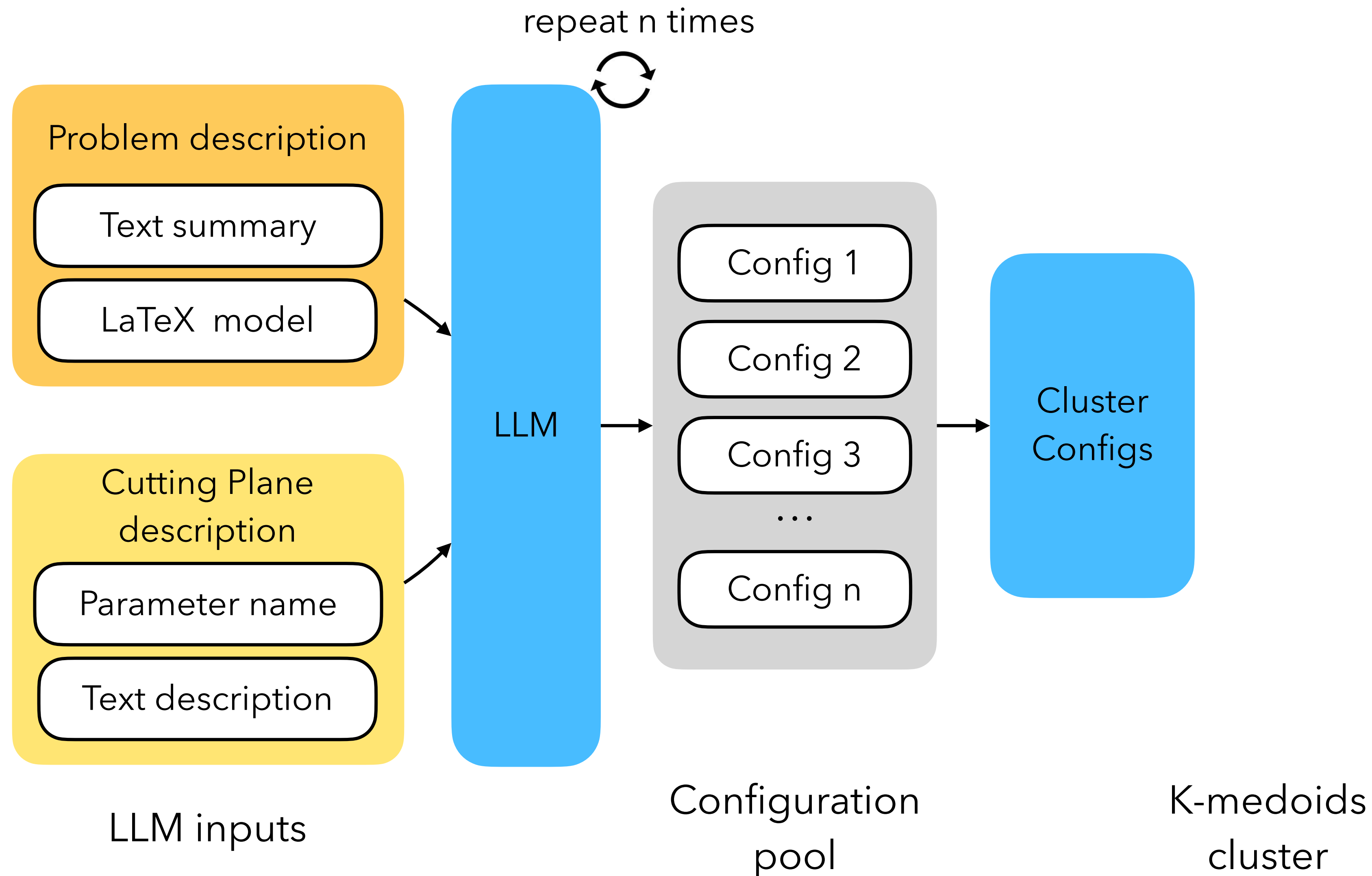
LLM for cold-start configuration pipeline



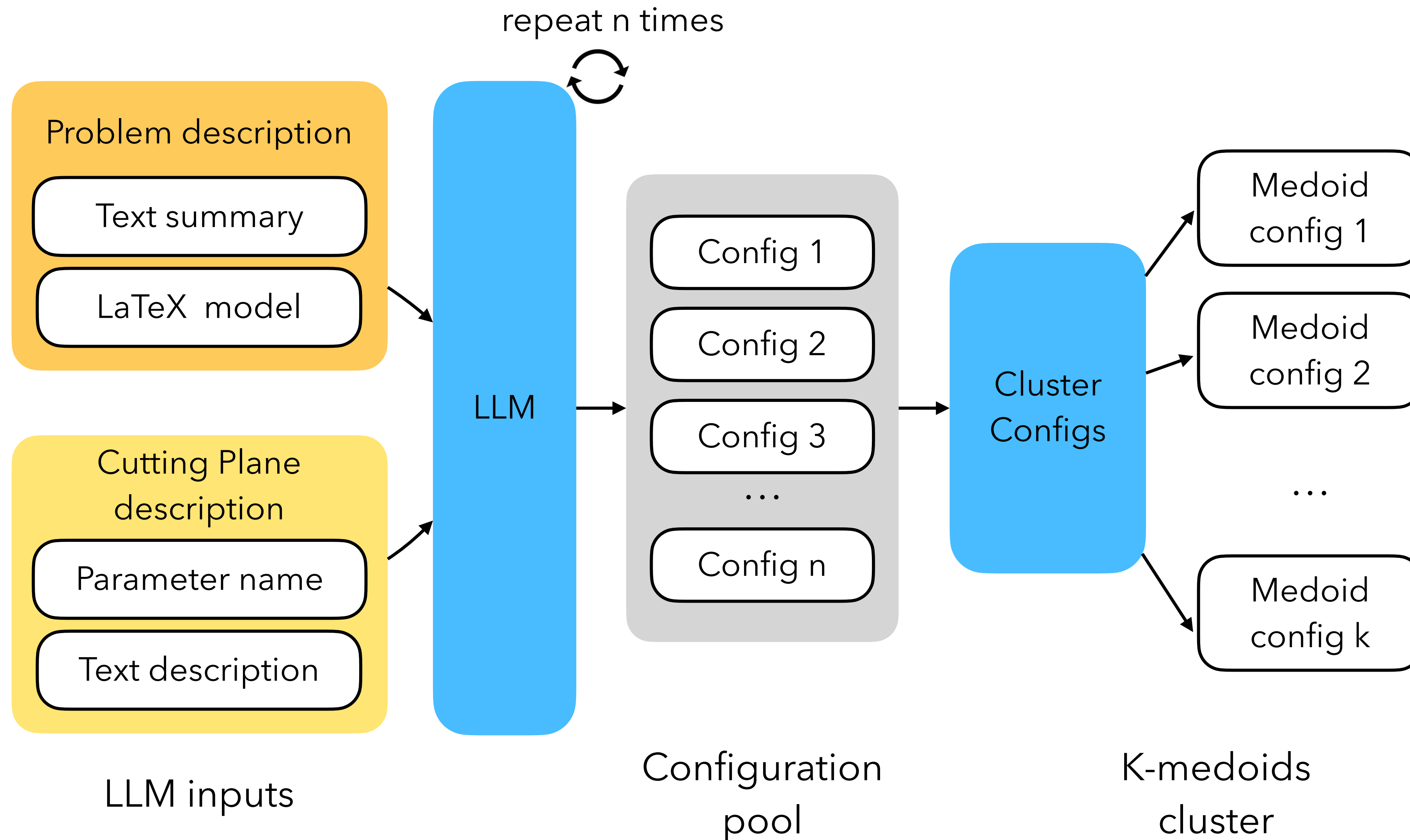
LLM for cold-start configuration pipeline



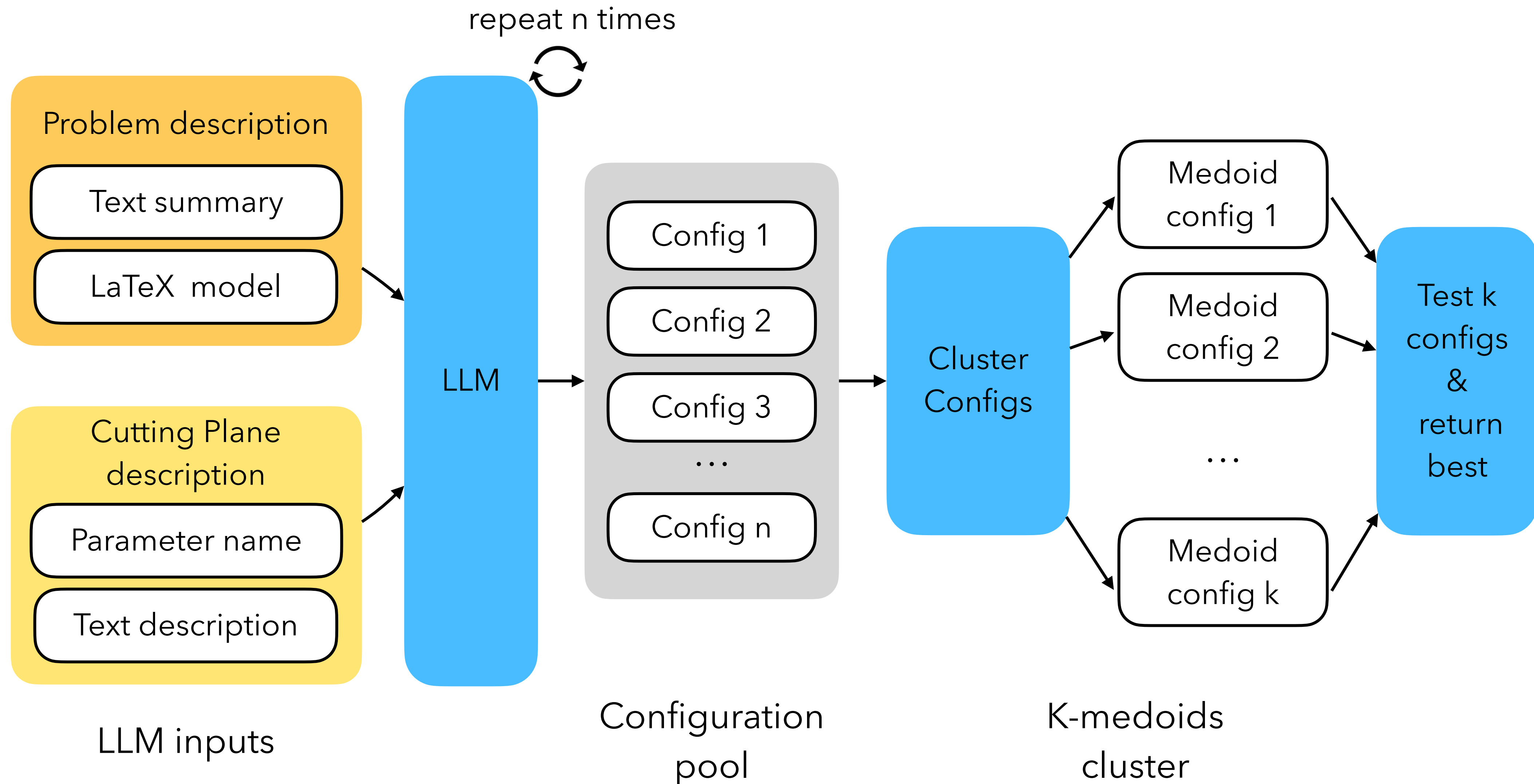
LLM for cold-start configuration pipeline



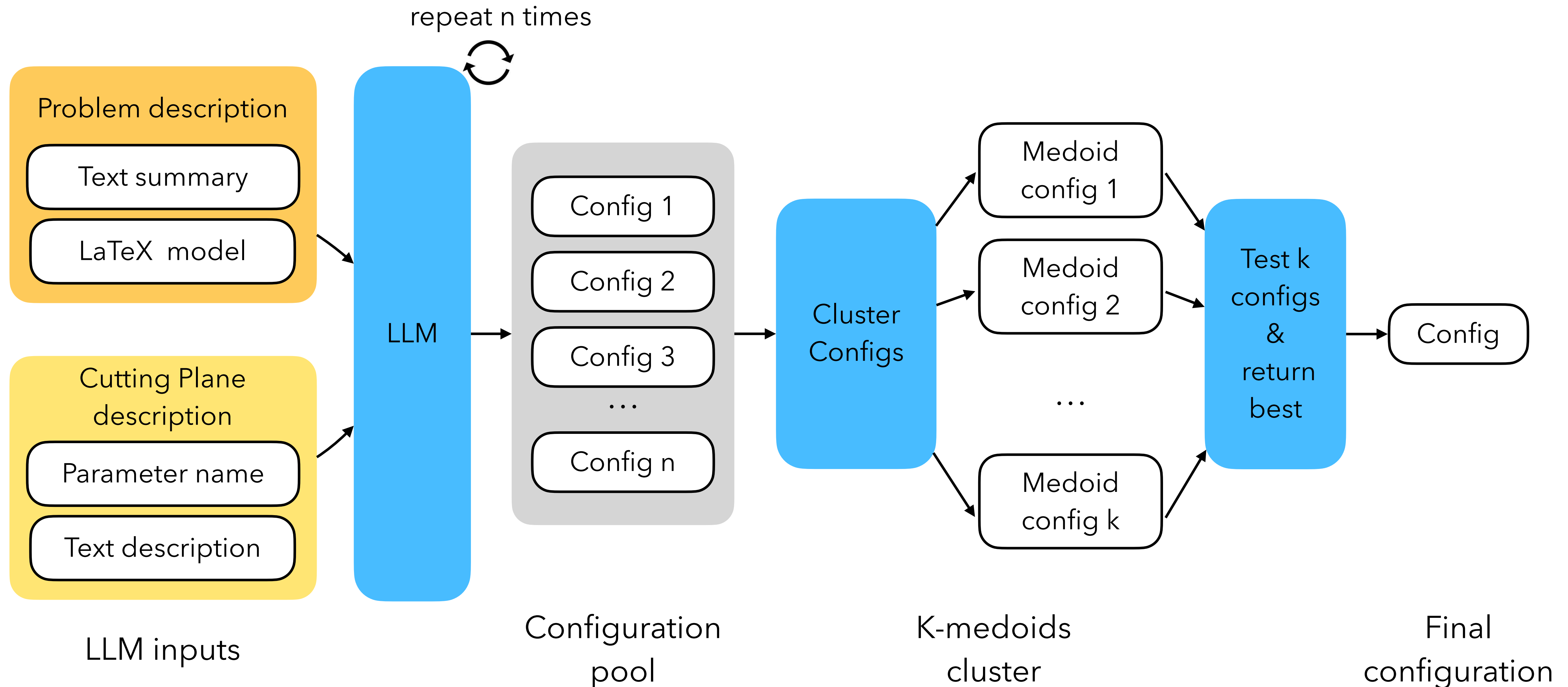
LLM for cold-start configuration pipeline



LLM for cold-start configuration pipeline



LLM for cold-start configuration pipeline



Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random
 - Use the one with best median performance on validation set

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random
 - Use the one with best median performance on validation set
- **Zero-shot**: use medoid of the largest cluster

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random
 - Use the one with best median performance on validation set
- **Zero-shot**: use medoid of the largest cluster
- **Cold-start(k)**:

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random
 - Use the one with best median performance on validation set
- **Zero-shot**: use medoid of the largest cluster
- **Cold-start(k)**:
 1. Run k medoids clustering

Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
 - Use the default setting for other CPs
- **Search(d)**: sample d candidate configurations uniformly at random
 - Use the one with best median performance on validation set
- **Zero-shot**: use medoid of the largest cluster
- **Cold-start(k)**:
 1. Run k medoids clustering
 2. Select the best performing medoid on the validation set

Experimental set-up

Datasets, model

Dataset	# vars	# constrs
Binary packing	300	300
Capacitated facility location	100	100
Combinatorial auction	100	500
Maximum independent set	500	1088
Max cut	54	134
Packing	60	60
Set cover	500	250
Load balancing	64340	61000
Middle-mile consolidation network design (MM)	569	248

Experimental set-up

Datasets, model

Dataset		# vars	# constrs
Classic MILP families	Binary packing	300	300
	Capacitated facility location	100	100
	Combinatorial auction	100	500
	Maximum independent set	500	1088
	Max cut	54	134
	Packing	60	60
Complex real-world MILP families	Set cover	500	250
	Load balancing	64340	61000
	Middle-mile consolidation network design (MM)	569	248

Experimental set-up

Datasets, model

	Dataset	# vars	# constrs
Classic MILP families	Binary packing	300	300
	Capacitated facility location	100	100
	Combinatorial auction	100	500
	Maximum independent set	500	1088
	Max cut	54	134
	Packing	60	60
	Set cover	500	250
Complex real-world MILP families	Load balancing	64340	61000
	Middle-mile consolidation network design (MM)	569	248

Model: GPT-4o

Training set size: 100

Val set size: 30

Evaluation metric:
% improvement over
default solve time

Empirical results

Cold-start(5) yields 6-71% faster runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

Empirical results

Cold-start(5) yields 6-71% faster runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

By testing only 5 configs,
we match/beat Search(500)
on **all instances**

Empirical results

Cold-start(5) yields **6-71% faster** runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

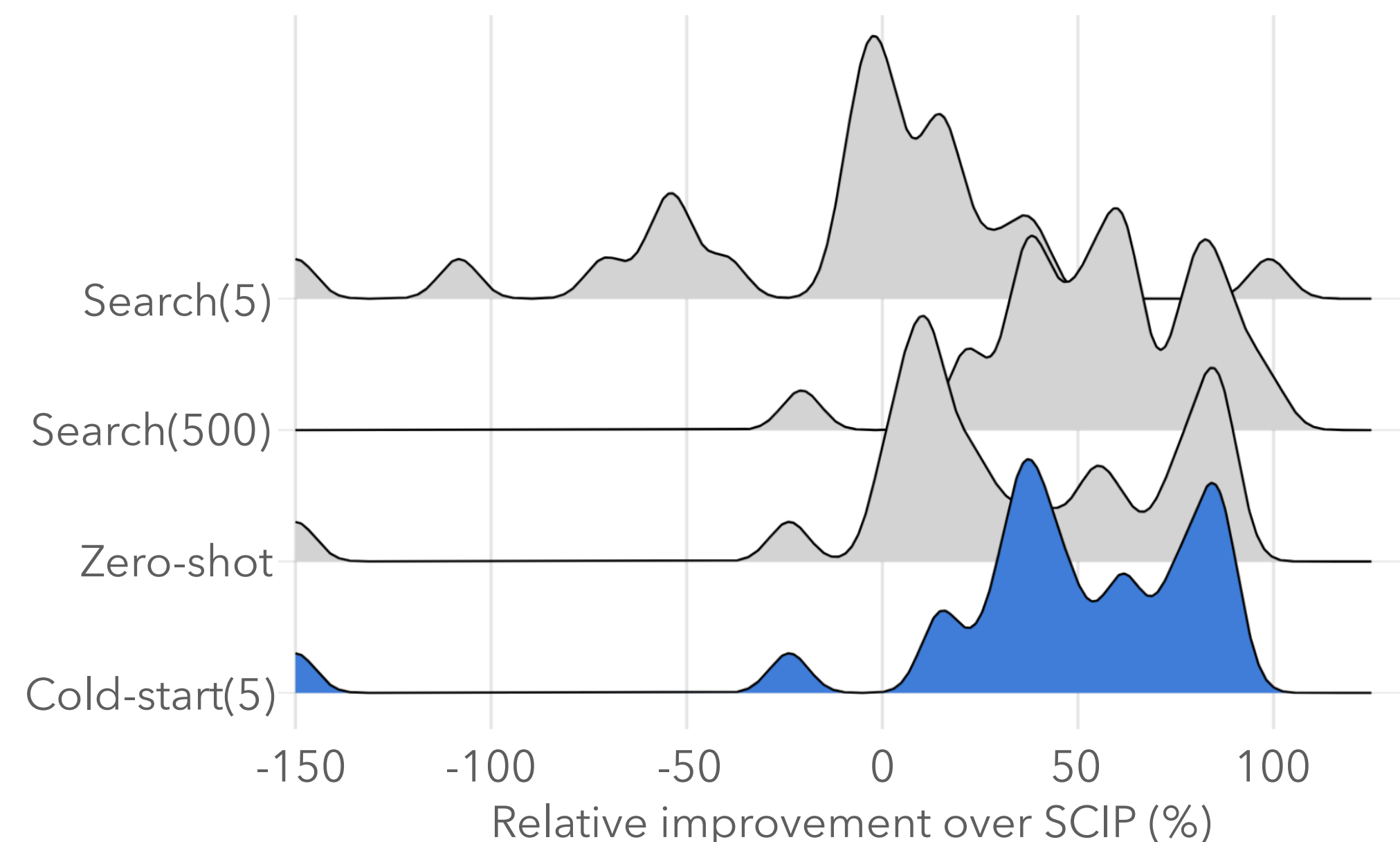
By testing only 5 configs,
we match/beat Search(500)
on **all instances**

Without solving **any** MILPs,
we match/beat Search(500)
on **4/9 instances**

On out-of-distribution instances

25 families of problem from MILP-Evolve dataset [Li et al. ICLR'25]

- New dataset, “evolving” pipeline for generating new MILP families
- Designed to be highly diverse to mimic real-world optimization scenarios



Ablations

Our design choices are robust

Setting	Ind. set	Max cut	Bin. pack.	MM
Ours	71.95	71.01	38.35	53.3

Ablations

Our design choices are robust

Setting	Ind. set	Max cut	Bin. pack.	MM
Ours	71.95	71.01	38.35	53.3
Disable cutting planes	-14.96	71.25	30.43	-150

Disabling CPs can reduce performance

Ablations

Our design choices are robust

Setting	Ind. set	Max cut	Bin. pack.	MM
Ours	71.95	71.01	38.35	53.3
Disable cutting planes	-14.96	71.25	30.43	-150
No CP text descr.	72.27	71.49	16.85	9.29

Disabling CPs can reduce performance

Our CP descriptions boost performance

Ablations

Our design choices are robust

Setting	Ind. set	Max cut	Bin. pack.	MM
Ours	71.95	71.01	38.35	53.3
Disable cutting planes	-14.96	71.25	30.43	-150
No CP text descr.	72.27	71.49	16.85	9.29
Ensembling strategies				
Average configuration	20.65	71.24	17.52	-11.08
Mode configuration	21.08	71.44	18.11	-12.63
Smallest configuration	20.83	70.91	17.42	-4.74

Disabling CPs can reduce performance

Our CP descriptions boost performance

k-medoids outperforms simpler heuristics

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**
- **Ensembling strategy** to build portfolio of high-performing configurations

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**
- **Ensembling strategy** to build portfolio of high-performing configurations
- Requires **no custom solver interface**

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**
- **Ensembling strategy** to build portfolio of high-performing configurations
- Requires **no custom solver interface**
- Competitive with existing configuration approaches

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**
- **Ensembling strategy** to build portfolio of high-performing configurations
- Requires **no custom solver interface**
- Competitive with existing configuration approaches
but only requires a **fraction of the training data and computation time**

Outline

1. Background
2. LLMs for speeding up solvers
 - i. Solver configuration
 - ii. Evolutionary search for MILP heuristics**
 - a. Background**
 - b. Cut selection
 - c. Large neighborhood search
 - d. Diving heuristics
3. Takeaways

LLMs for automated heuristic design

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
 - Fisher, Thompson, '63; Crowston et al., '63; survey by Burke et al., JORS'13

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
 - Fisher, Thompson, '63; Crowston et al., '63; survey by Burke et al., JORS'13
- Automated heuristic design with LLMs:

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
 - Fisher, Thompson, '63; Crowston et al., '63; survey by Burke et al., JORS'13
- Automated heuristic design with LLMs:
 - LLMs generate heuristic code

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
 - Fisher, Thompson, '63; Crowston et al., '63; survey by Burke et al., JORS'13
- Automated heuristic design with LLMs:
 - LLMs generate heuristic code
 - Heuristic **fitness** scored automatically

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
 - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
 - Fisher, Thompson, '63; Crowston et al., '63; survey by Burke et al., JORS'13
- Automated heuristic design with LLMs:
 - LLMs generate heuristic code
 - Heuristic **fitness** scored automatically
 - **Evolutionary selection** keeps improved heuristics

E.g., FunSearch [Romera-Paredes, Nature'24], ReEvo [Ye et al., NeurIPS'24], AlphaEvolve [Novikov et al., '25], ...

Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins



Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins
- Items arrive sequentially



Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins
- Items arrive sequentially
 - Must be packed into a bin immediately



Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins
- Items arrive sequentially
 - Must be packed into a bin immediately
 - No knowledge of future arrivals



Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins
- Items arrive sequentially
 - Must be packed into a bin immediately
 - No knowledge of future arrivals
- Each bin has fixed capacity (experiments: $C = 100$)



Figure by Fawzi, Romera-Paredes

Example: Online bin packing

Heuristic representation

Heuristic

Example: Online bin packing

Heuristic representation

Heuristic

Code

```
def heuristic(item, bins):  
    """  
    item: scalar item size  
    bins: 1D np.array of remaining capacities  
    returns: per-bin scores (higher is better)  
    ...
```

Example: Online bin packing

Heuristic representation

Heuristic

Code

```
def heuristic(item, bins):  
    """  
    item: scalar item size  
    bins: 1D np.array of remaining capacities  
    returns: per-bin scores (higher is better)  
    ...
```

Natural language description

Incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.

Example: Online bin packing

Heuristic representation

Heuristic

Code

```
def heuristic(item, bins):  
    """  
    item: scalar item size  
    bins: 1D np.array of remaining capacities  
    returns: per-bin scores (higher is better)  
    ...
```

Natural language description

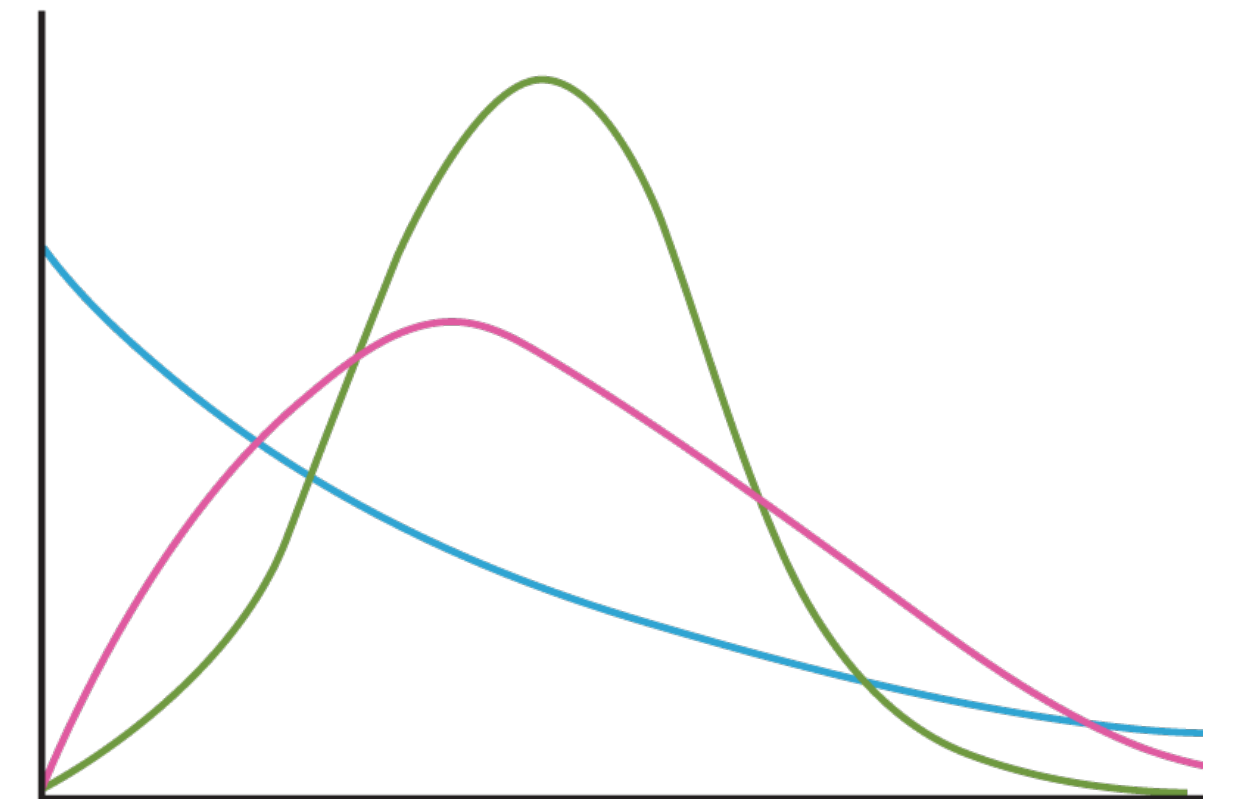
Incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.

Fitness: 0.0196

Example: Online bin packing

Fitness metric

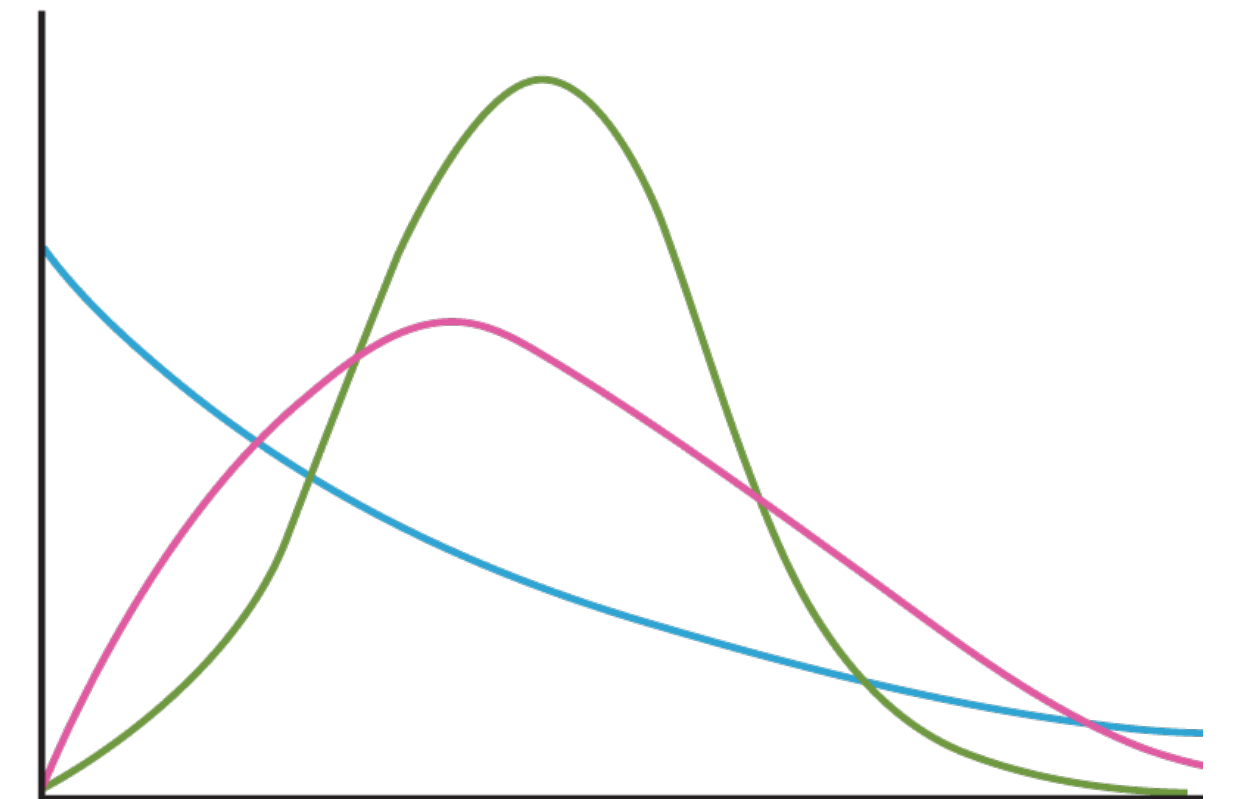
- **Test instances** [Romera-Paredes et al., Nature'24]:
 - Five Weibull test instances, each with 5000 items



Example: Online bin packing

Fitness metric

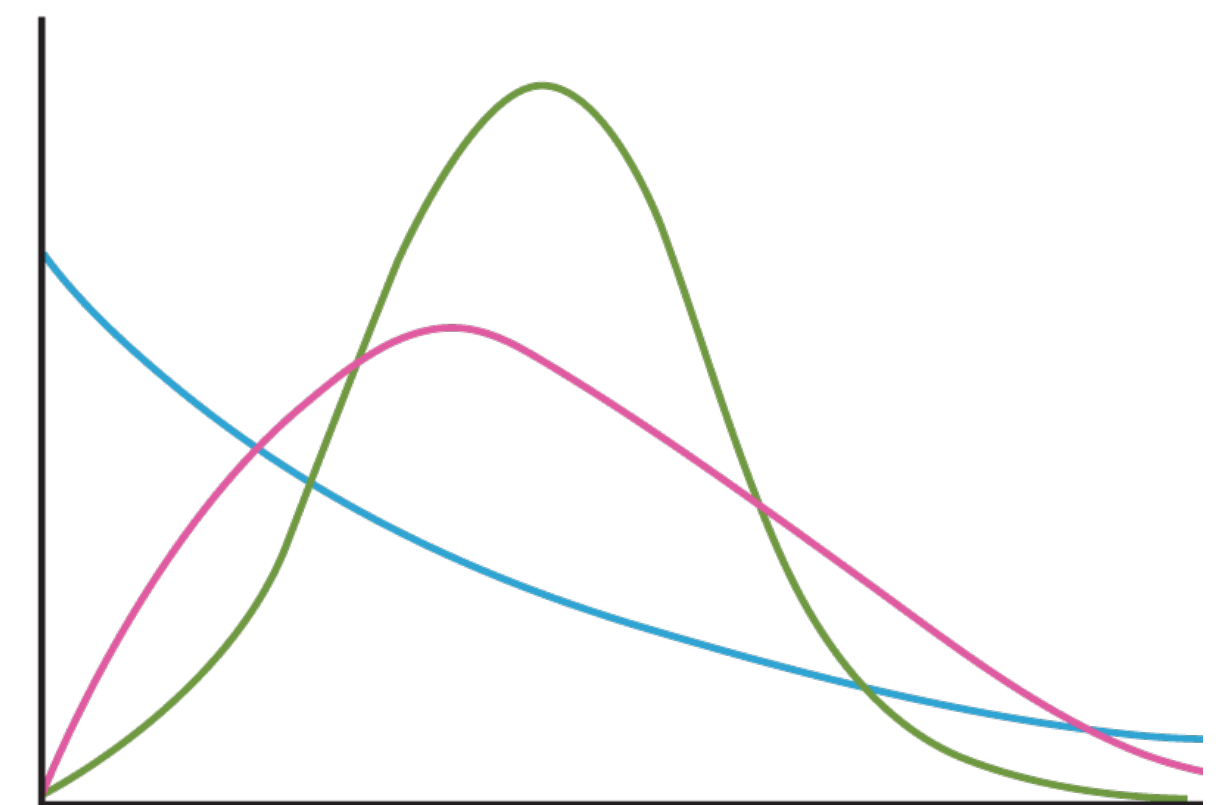
- **Test instances** [Romera-Paredes et al., Nature'24]:
 - Five Weibull test instances, each with 5000 items
- ℓb = lower bound on opt bin count [Martello & Toth '90]



Example: Online bin packing

Fitness metric

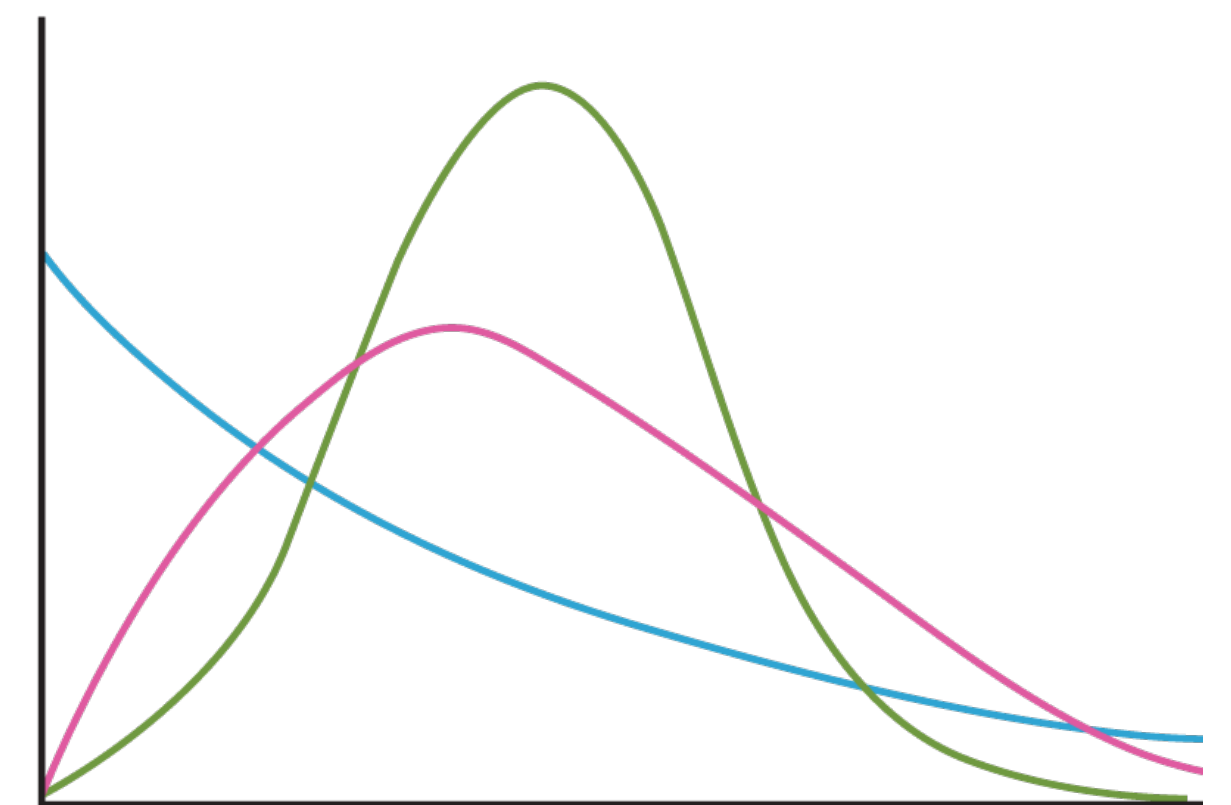
- **Test instances** [Romera-Paredes et al., Nature'24]:
 - Five Weibull test instances, each with 5000 items
- ℓb = lower bound on opt bin count [Martello & Toth '90]
- n = number of bins used by heuristic



Example: Online bin packing

Fitness metric

- **Test instances** [Romera-Paredes et al., Nature'24]:
 - Five Weibull test instances, each with 5000 items
- ℓb = lower bound on opt bin count [Martello & Toth '90]
- n = number of bins used by heuristic
- Fitness = $\text{avg} \left(\frac{\ell b}{n} \right)$ across the test instances



Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*



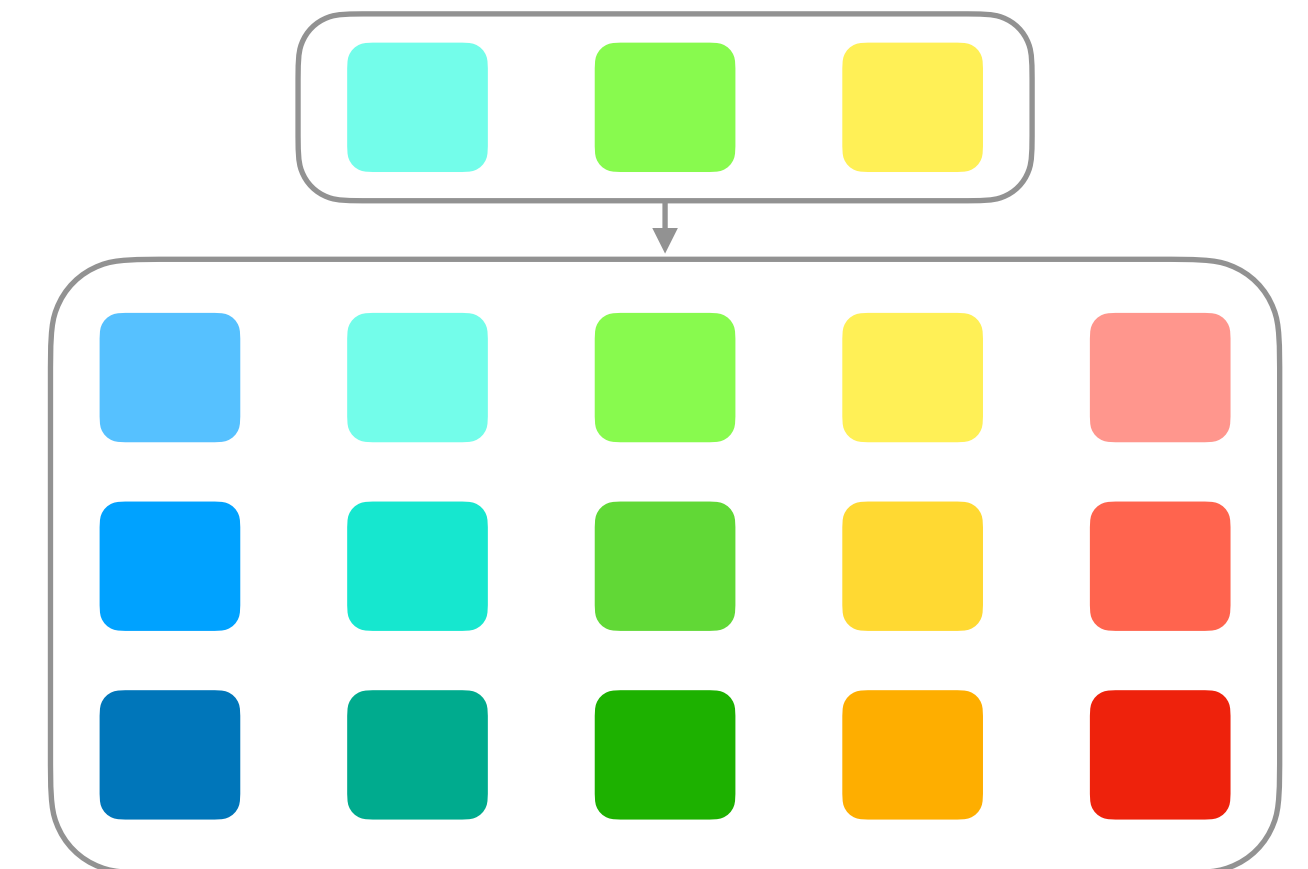
Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*
2. **Heuristic generation:**



Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*
2. **Heuristic generation:**
Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)



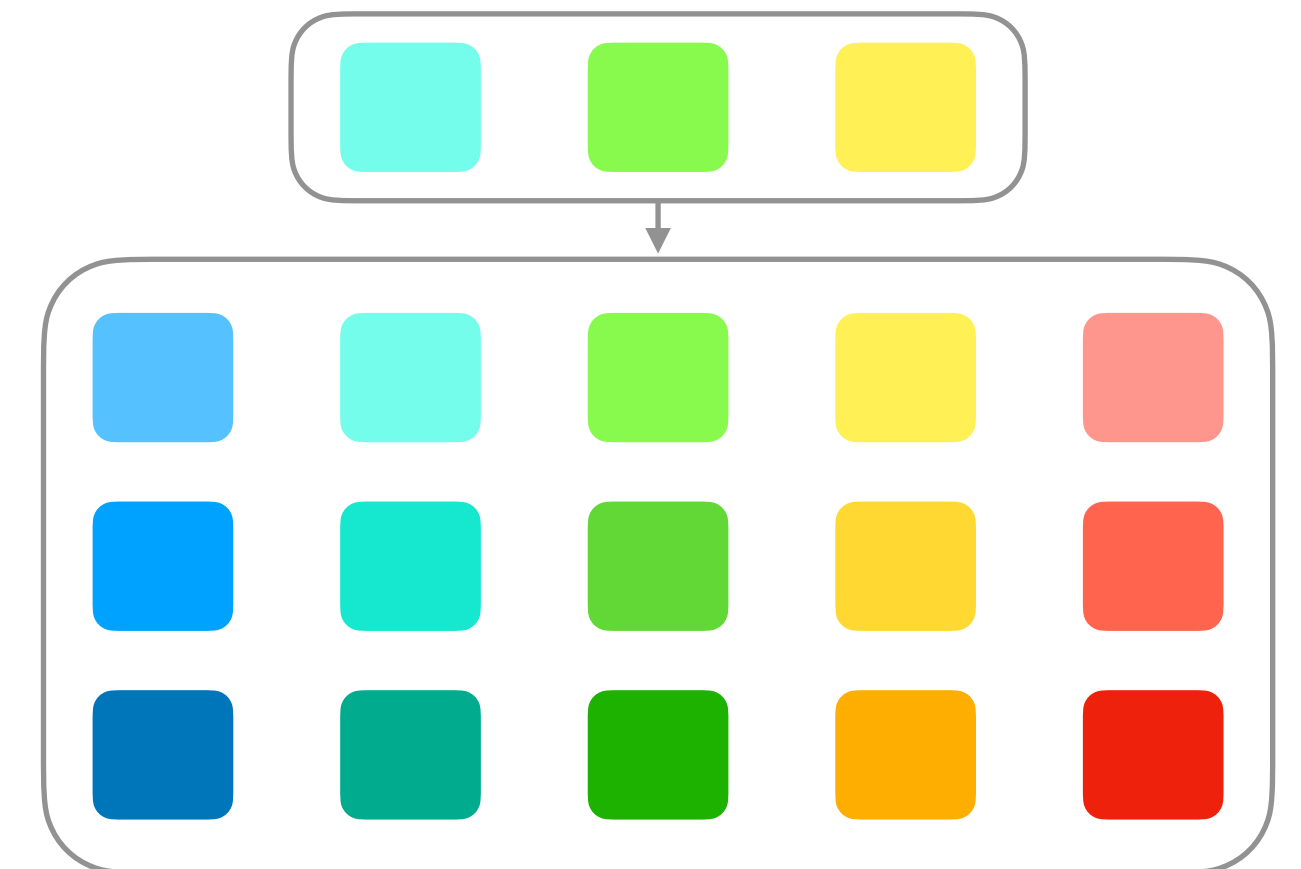
Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*

2. **Heuristic generation:**

Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

i. Select parent heuristic(s) to form prompt



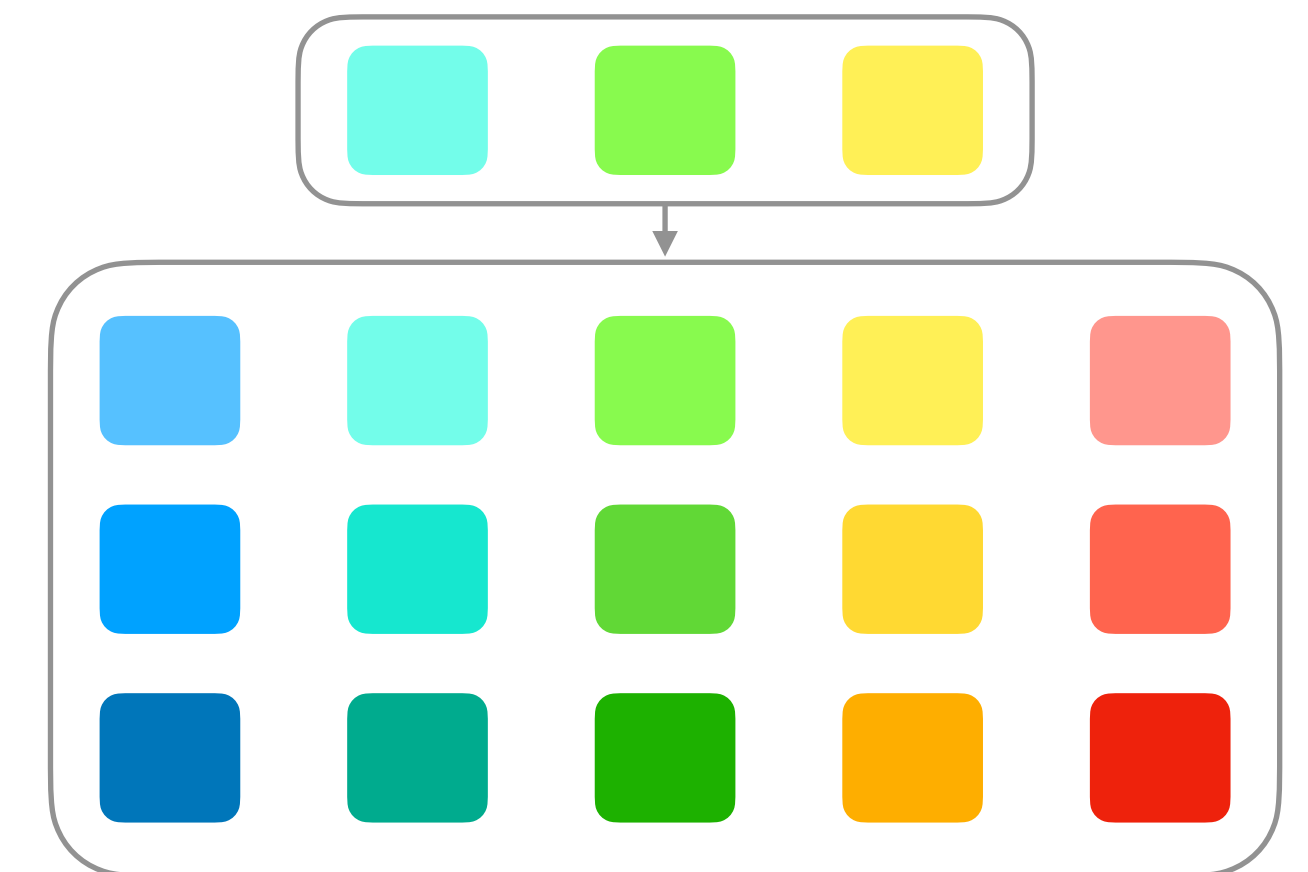
Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*

2. **Heuristic generation:**

Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

- i. Select parent heuristic(s) to form prompt
- ii. LLM generates new thought and code



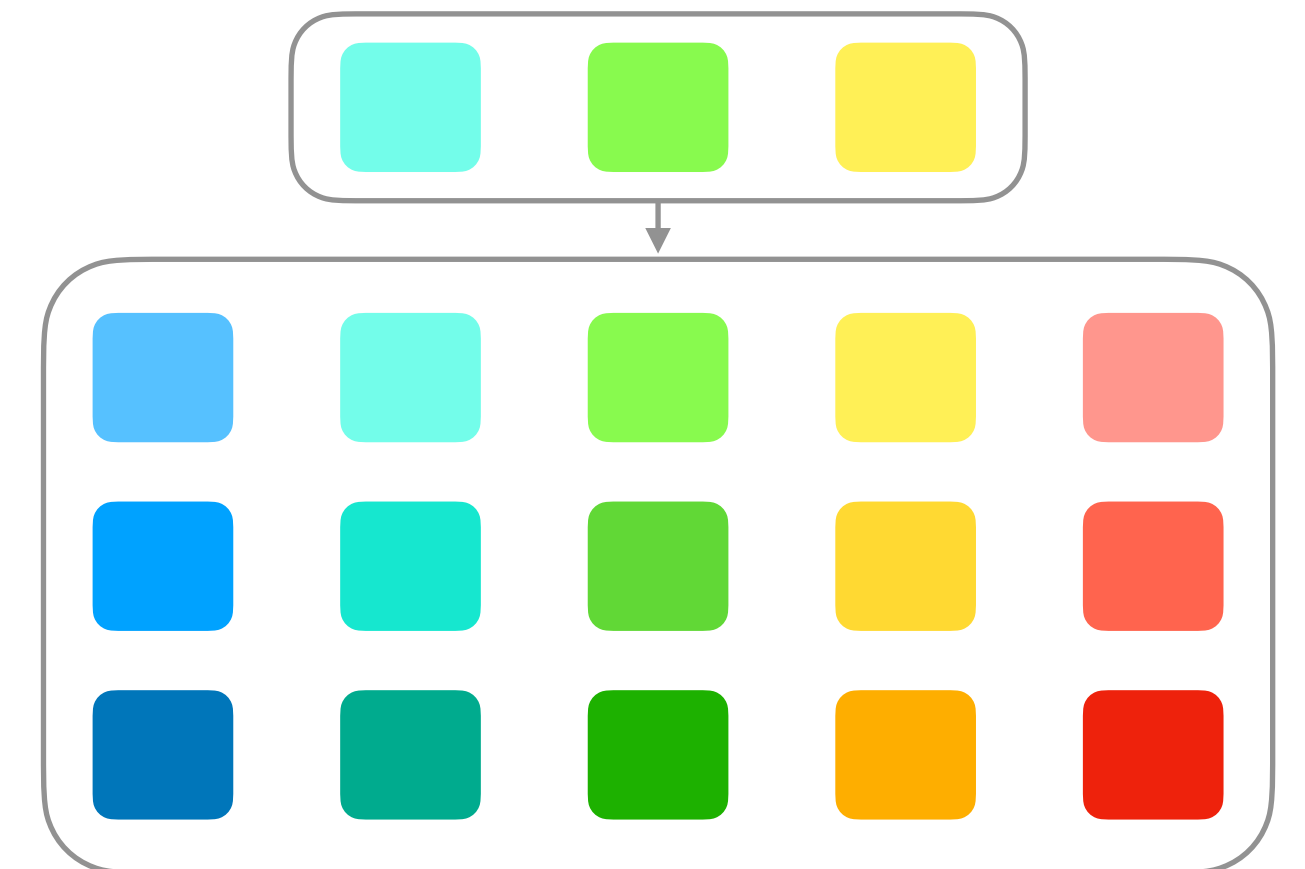
Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*

2. **Heuristic generation:**

Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

- i. Select parent heuristic(s) to form prompt
- ii. LLM generates new thought and code
- iii. Evaluate fitness on test instances



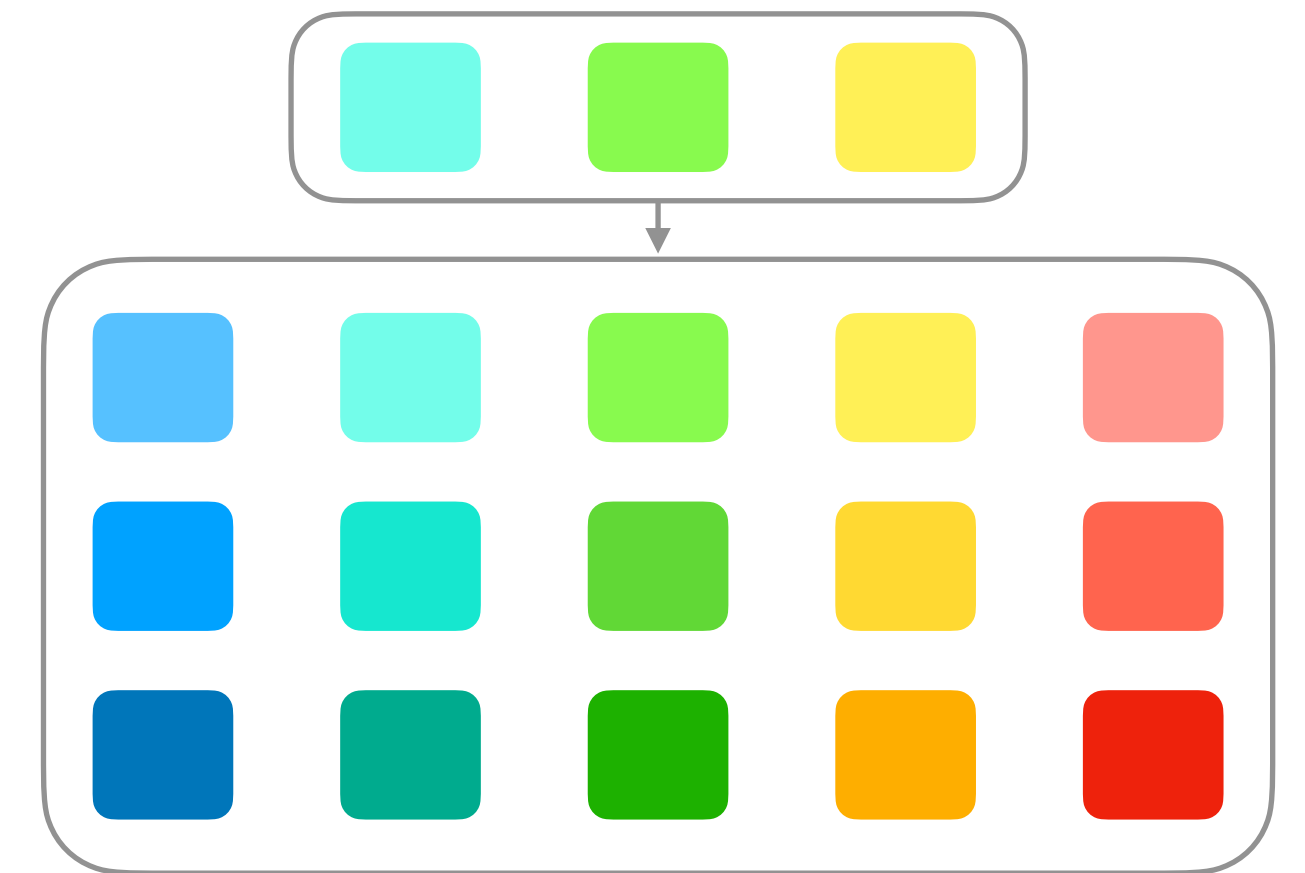
Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*

2. **Heuristic generation:**

Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

- i. Select parent heuristic(s) to form prompt
- ii. LLM generates new thought and code
- iii. Evaluate fitness on test instances
- iv. Add feasible heuristics to population



Algorithm overview

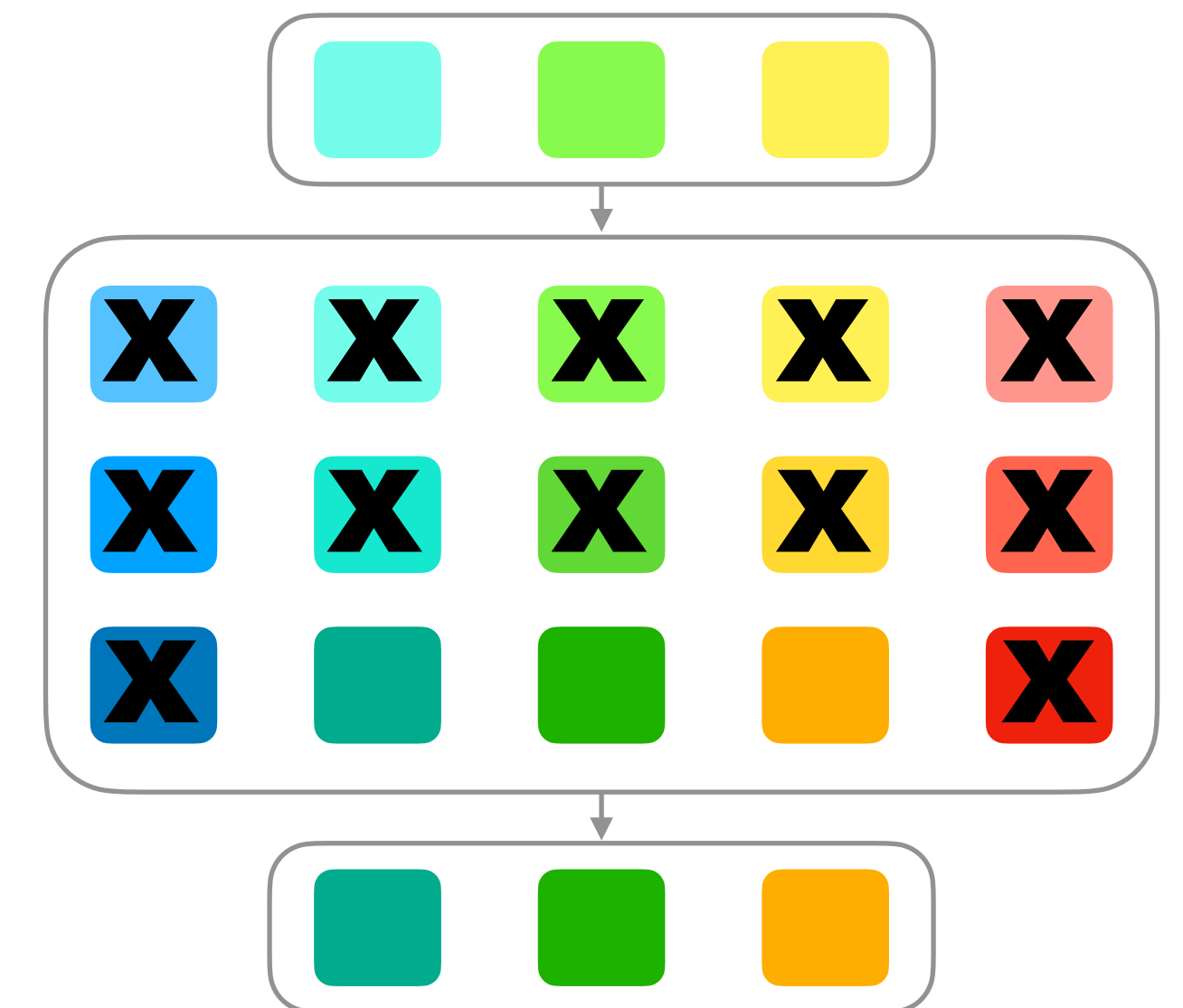
1. **Initialization:** generate N initial heuristics using *Initialization Prompt*

2. **Heuristic generation:**

Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

- Select parent heuristic(s) to form prompt
- LLM generates new thought and code
- Evaluate fitness on test instances
- Add feasible heuristics to population

3. **Retain** top N heuristics by fitness; return to Step 1



Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item

Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score

Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score
- If a bin is full, it will not be used

Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score
- If a bin is full, it will not be used
- The final goal is to minimize the number of used bins

Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score
- If a bin is full, it will not be used
- The final goal is to minimize the number of used bins
- Firstly, describe your new heuristic and main steps in one sentence

Example: Online bin packing

Initialization prompt

- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score
- If a bin is full, it will not be used
- The final goal is to minimize the number of used bins
- Firstly, describe your new heuristic and main steps in one sentence

Evolution prompts

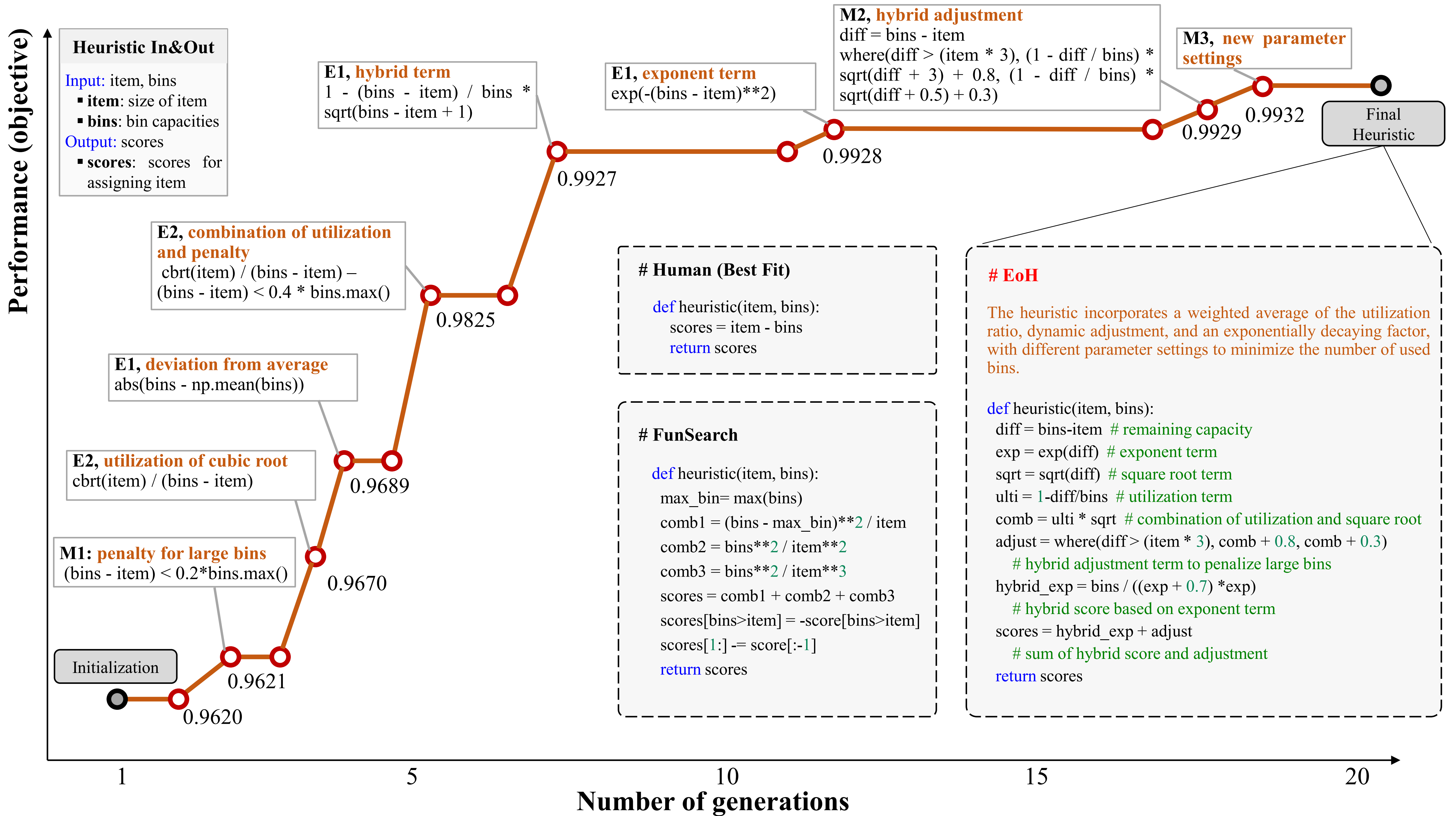
- **E1 - Diverse exploration:** generate entirely new heuristic ideas from scratch
- **E2 - Shared-Idea variants:** generate based on high-performing “themes”

Evolution prompts

- **E1 - Diverse exploration:** generate entirely new heuristic ideas from scratch
- **E2 - Shared-Idea variants:** generate based on high-performing “themes”
- **M1 - Edit:** modify an existing heuristic
- **M2 - Parameter tuning:** fine-tune numeric settings or thresholds in code

Evolution prompts

- **E1 - Diverse exploration:** generate entirely new heuristic ideas from scratch
- **E2 - Shared-Idea variants:** generate based on high-performing “themes”
- **M1 - Edit:** modify an existing heuristic
- **M2 - Parameter tuning:** fine-tune numeric settings or thresholds in code
- **M3 - simplification:** prune unnecessary components or redundant logic



Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits

Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits
 - Best Fit: place item in bin w/ least available space

Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits
 - Best Fit: place item in bin w/ least available space
 - Published FunSearch heuristic as-is

Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits
 - Best Fit: place item in bin w/ least available space
 - Published FunSearch heuristic as-is
- **Problem sizes:** 1000–10,000 items

Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits
 - Best Fit: place item in bin w/ least available space
 - Published FunSearch heuristic as-is
- **Problem sizes:** 1000–10,000 items
- **Capacities:** $C = 100$ and $C = 500$

Example: Online bin packing

Experimental setup

- **Baselines:**
 - First Fit: place item in first bin that fits
 - Best Fit: place item in bin w/ least available space
 - Published FunSearch heuristic as-is
- **Problem sizes:** 1000–10,000 items
- **Capacities:** $C = 100$ and $C = 500$
- **Each setting:** 5 randomly generated instances

Bin packing results

		C=100				C=500		
Method	1k items	5k items	10k items		1k items	5k items	10k items	
First Fit	5.32%	4.40%	4.44%		4.97%	4.27%	4.28%	
Best Fit	4.87%	4.08%	4.09%		4.50%	3.91%	3.95%	
FunSearch	3.78%	0.80%	0.33%		6.75%	1.47%	0.74%	
EoH	2.24%	0.80%	0.61%		2.13%	0.78%	0.61%	

Metric: average gap to lower bound [Martello & Toth '90]

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
 - Evolves Pareto-optimal heuristics for multi-objective optimization

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
 - Evolves Pareto-optimal heuristics for multi-objective optimization
- **AlphaEvolve** [Novikov et al., arXiv'25]

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
 - Evolves Pareto-optimal heuristics for multi-objective optimization
- **AlphaEvolve** [Novikov et al., arXiv'25]
 - Scales to file-level evolution

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
 - Evolves Pareto-optimal heuristics for multi-objective optimization
- **AlphaEvolve** [Novikov et al., arXiv'25]
 - Scales to file-level evolution
 - Open-source version: OpenEvolve [Sharma, GitHub'25]

Beyond EoH

Recent examples

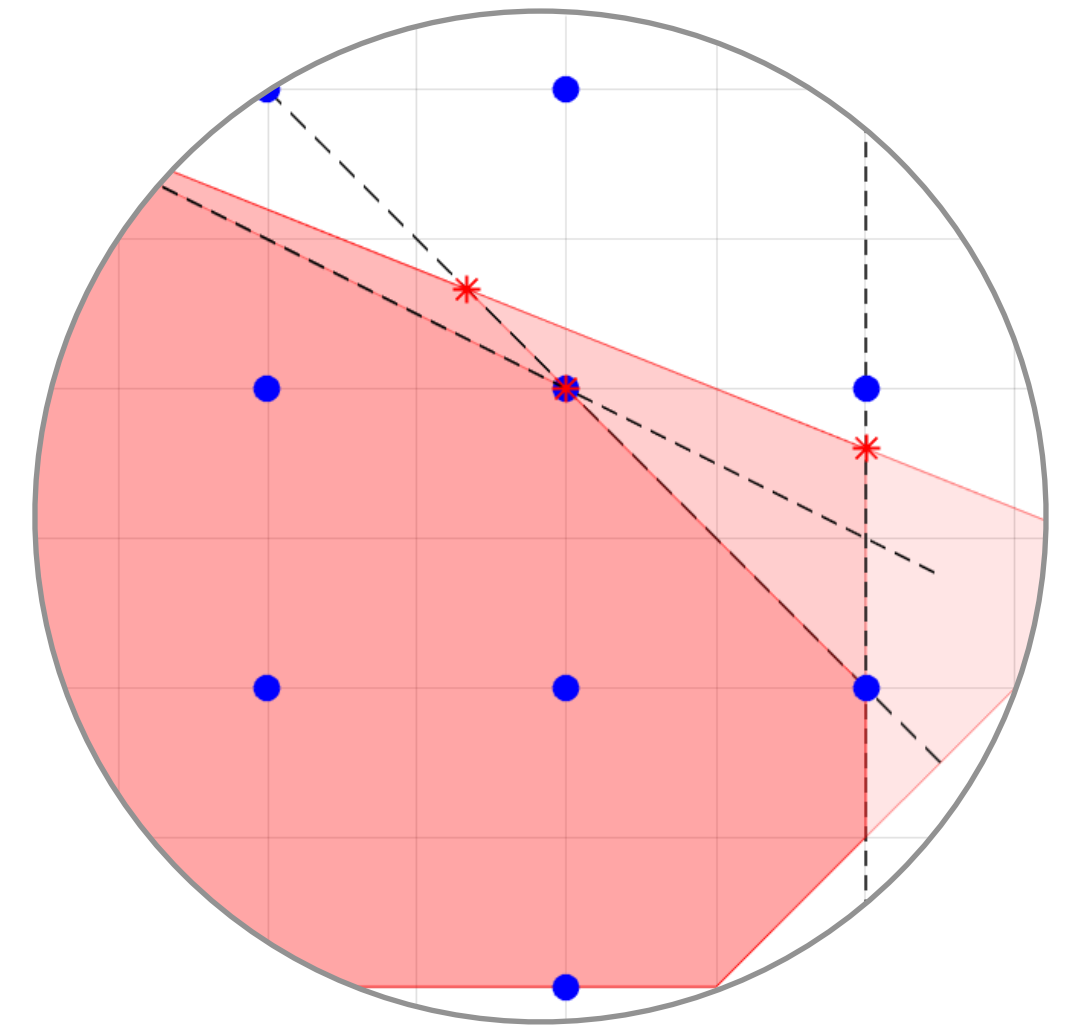
- **Reflective Evolution** [Ye et al., NeurIPS'24]
 - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
 - Evolves Pareto-optimal heuristics for multi-objective optimization
- **AlphaEvolve** [Novikov et al., arXiv'25]
 - Scales to file-level evolution
 - Open-source version: OpenEvolve [Sharma, GitHub'25]
 - Integrate with Deep Research methods: DeepEvolve [Liu et al., arXiv'25]

Outline

1. Background
2. LLMs for speeding up solvers
 - i. Solver configuration
 - ii. Evolutionary search for MILP heuristics
 - a. Background
 - b. Cut selection**
 - c. Large neighborhood search
 - d. Diving heuristics
3. Takeaways

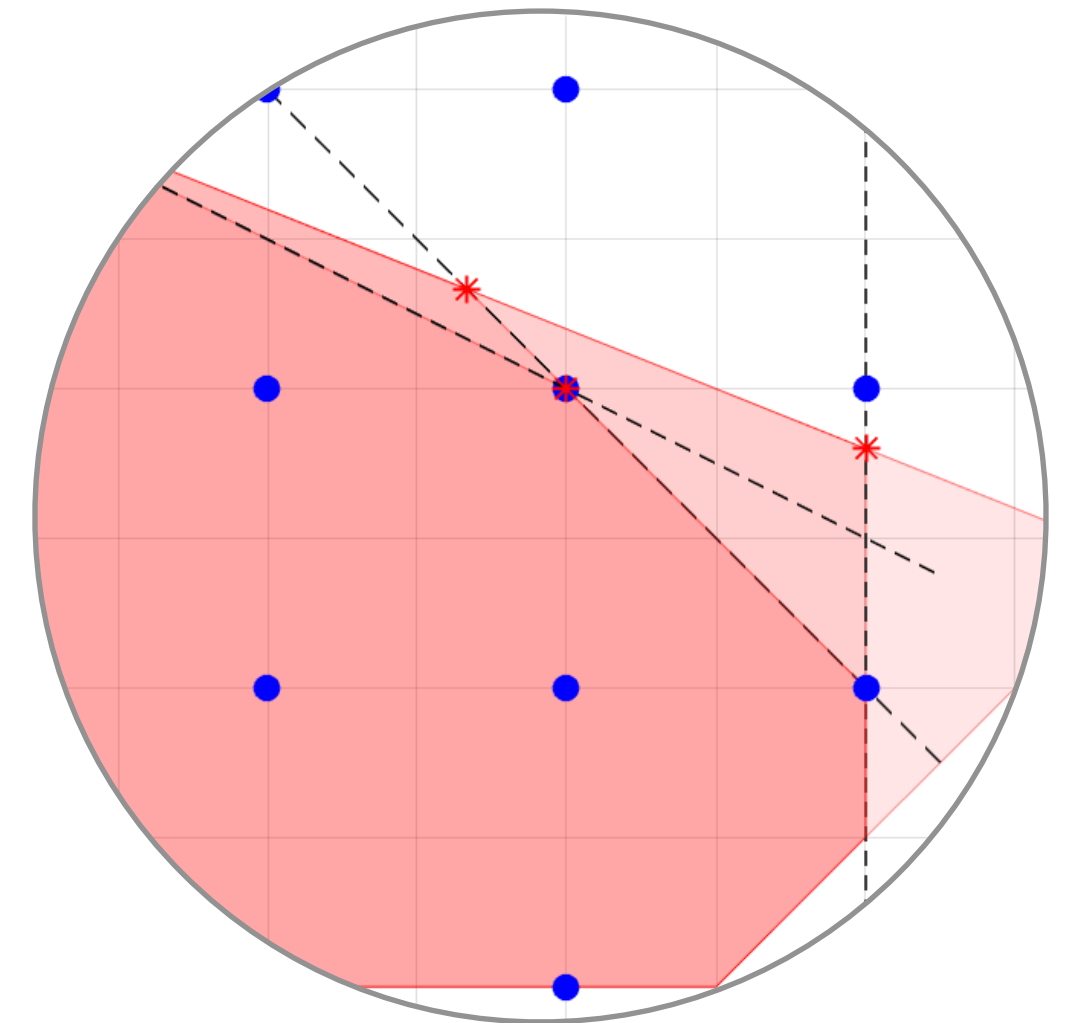
Overview of approach

- **EvoCut** automates cut discovery using LLMs plus evolution



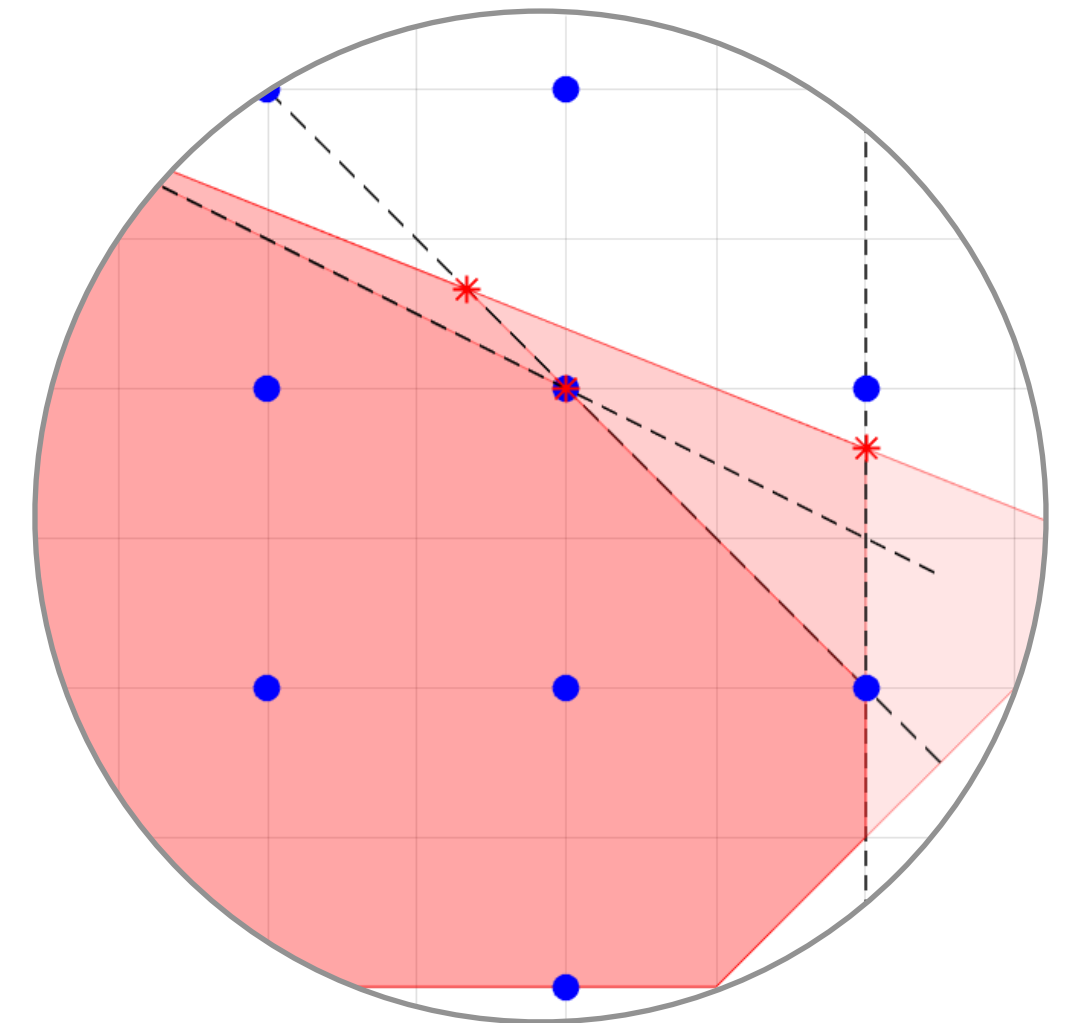
Overview of approach

- **EvoCut** automates cut discovery using LLMs plus evolution
 - Initializes cut population; evolves via crossover and mutation



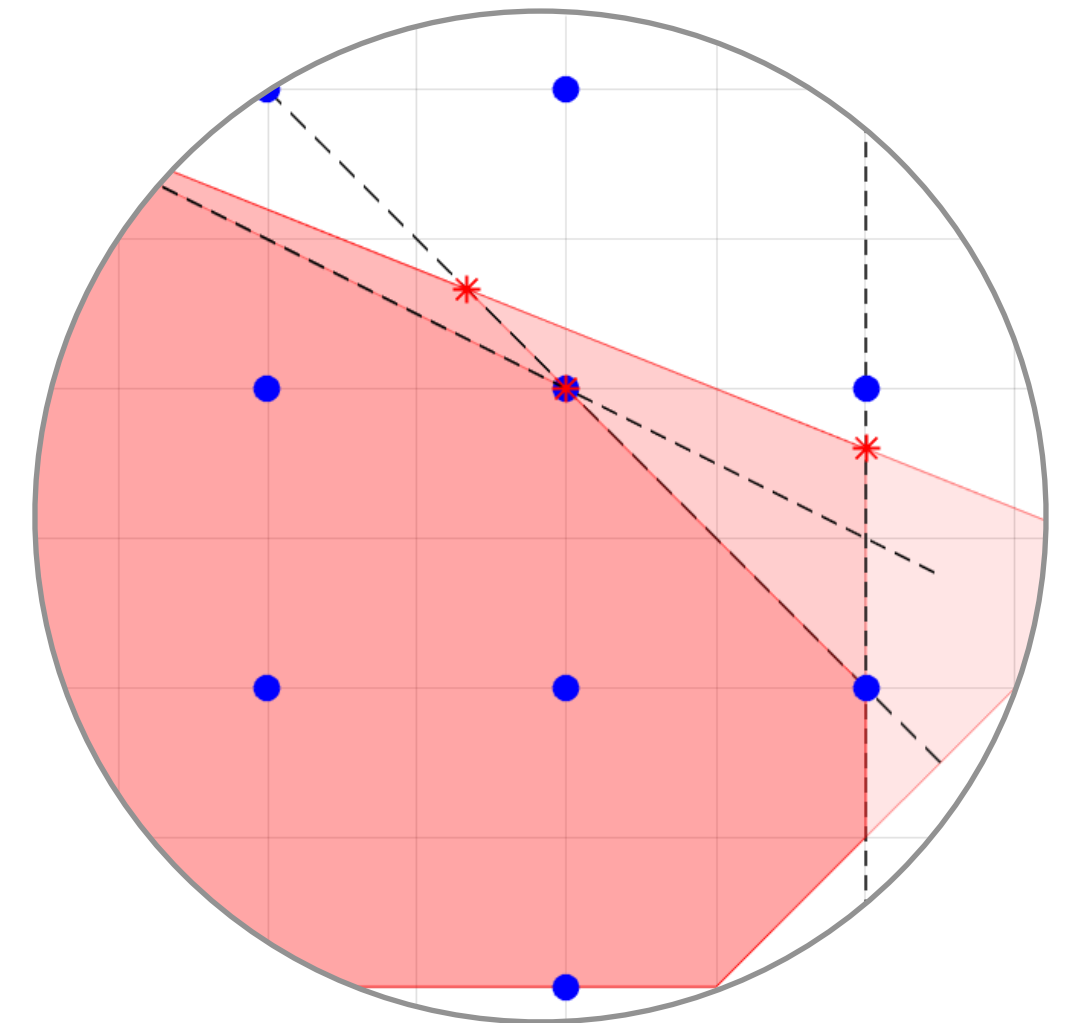
Overview of approach

- **EvoCut** automates cut discovery using LLMs plus evolution
 - Initializes cut population; evolves via crossover and mutation
- Empirically checks **optimal-solution preservation** and fractional **separation**



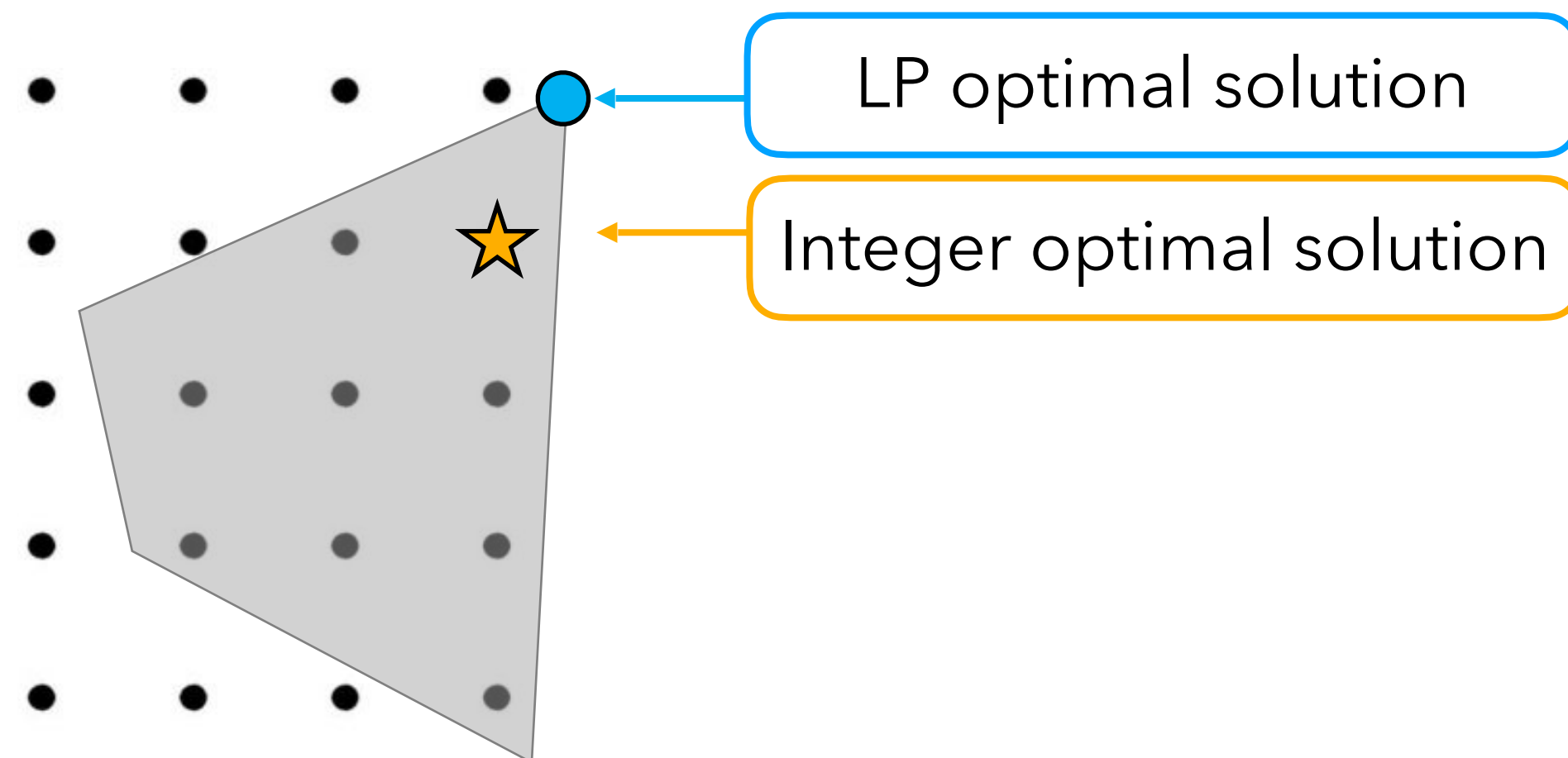
Overview of approach

- **EvoCut** automates cut discovery using LLMs plus evolution
 - Initializes cut population; evolves via crossover and mutation
- Empirically checks **optimal-solution preservation** and fractional **separation**
- Scores cuts by relative **optimality-gap reduction**



Acceleration cuts

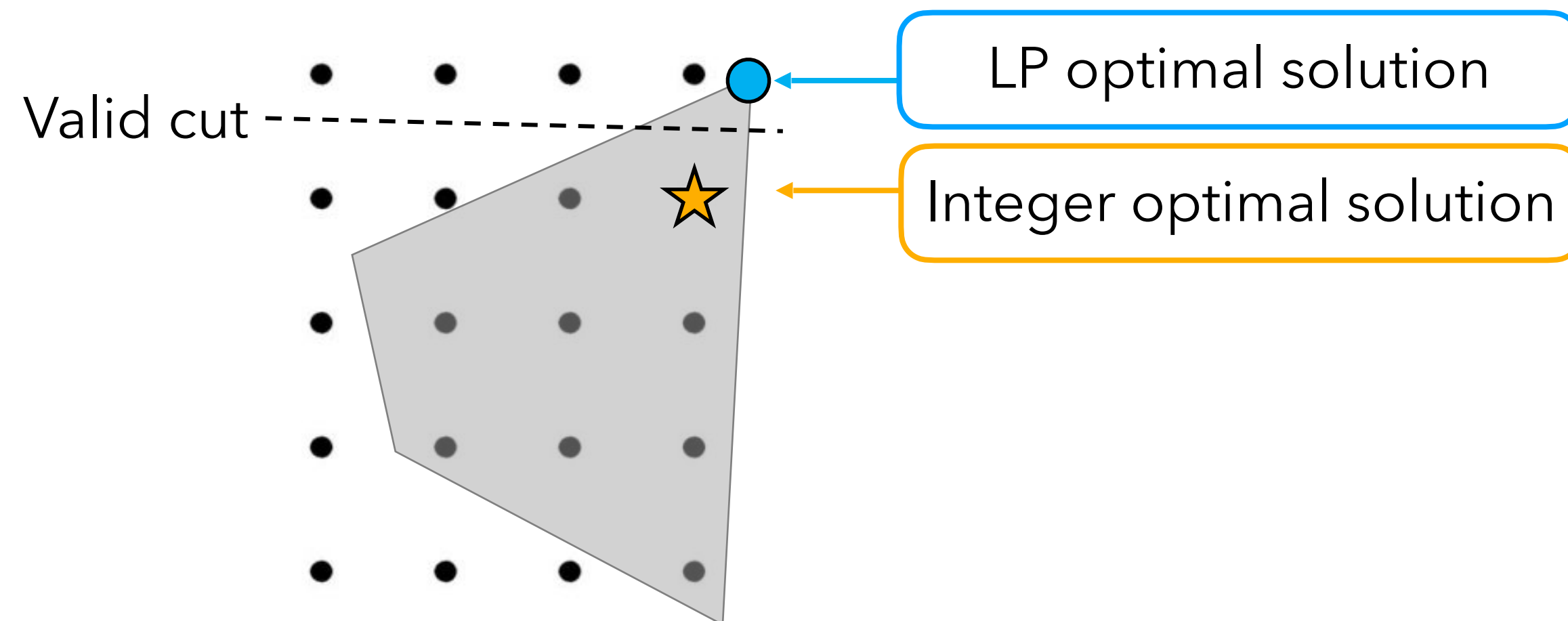
Inequalities added to speed up MILP solving



Acceleration cuts

Inequalities added to speed up MILP solving

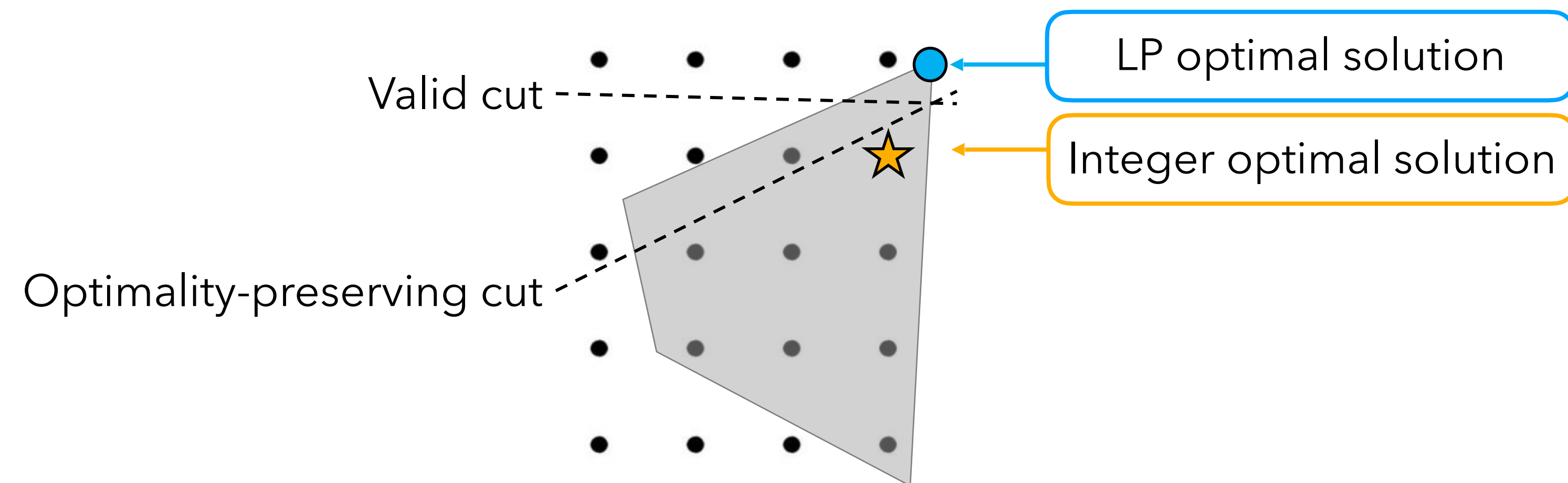
- **Valid cut:** doesn't separate any integer-feasible point



Acceleration cuts

Inequalities added to speed up MILP solving

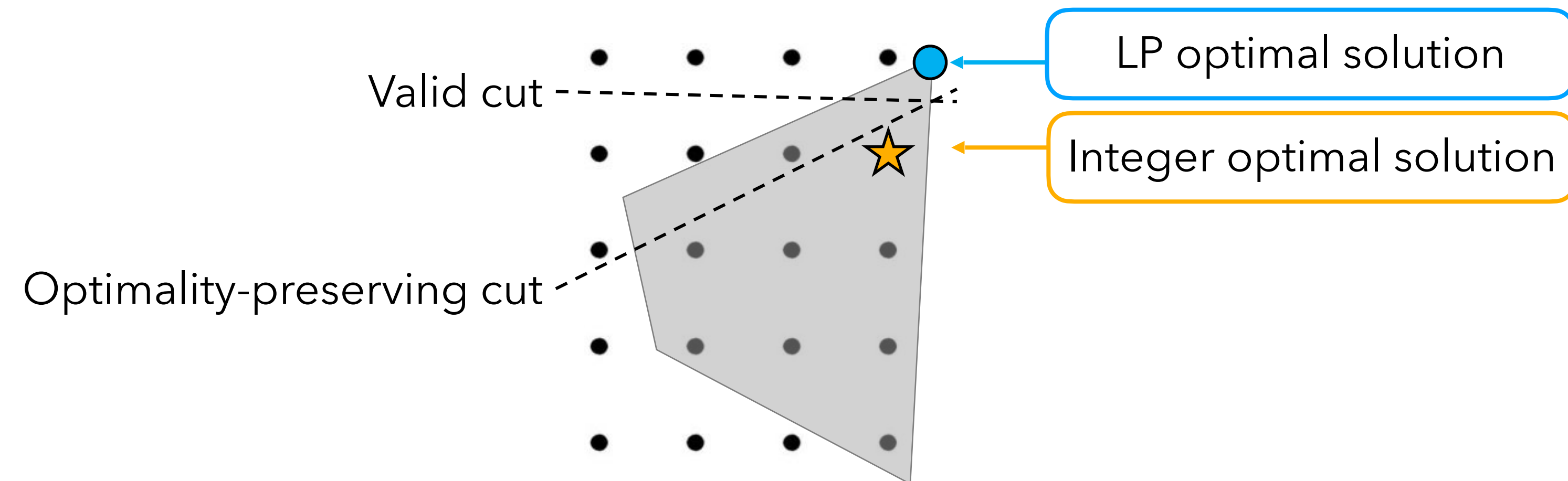
- **Valid cut:** doesn't separate any integer-feasible point
- **Optimality-preserving cut:** doesn't separate the opt integer-feasible point



Acceleration cuts

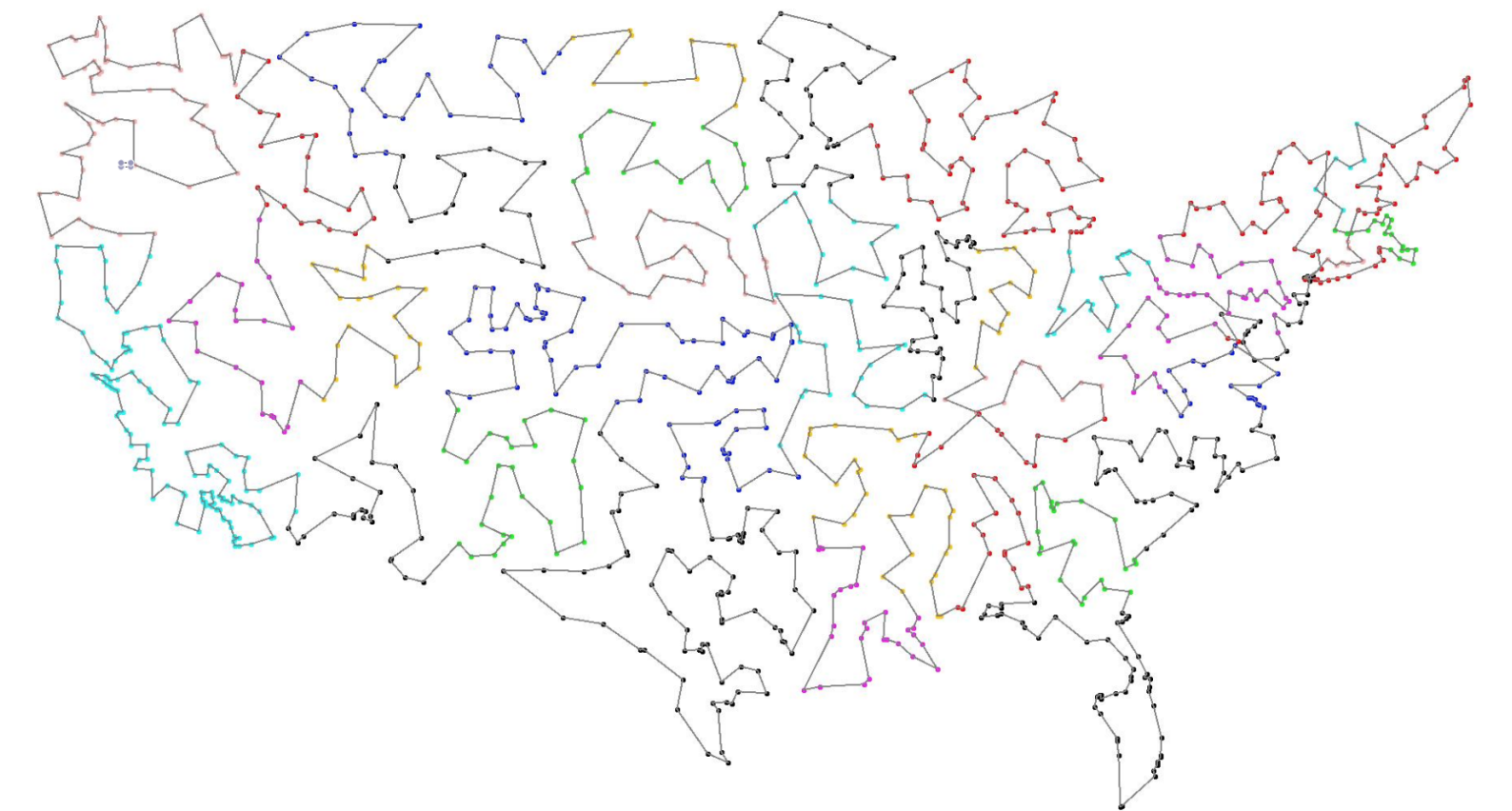
Inequalities added to speed up MILP solving

- **Valid cut:** doesn't separate any integer-feasible point
- **Optimality-preserving cut:** doesn't separate the opt integer-feasible point
- EvoCut cuts aren't proven optimality-preserving; empirically checked



Example: Traveling salesman problem

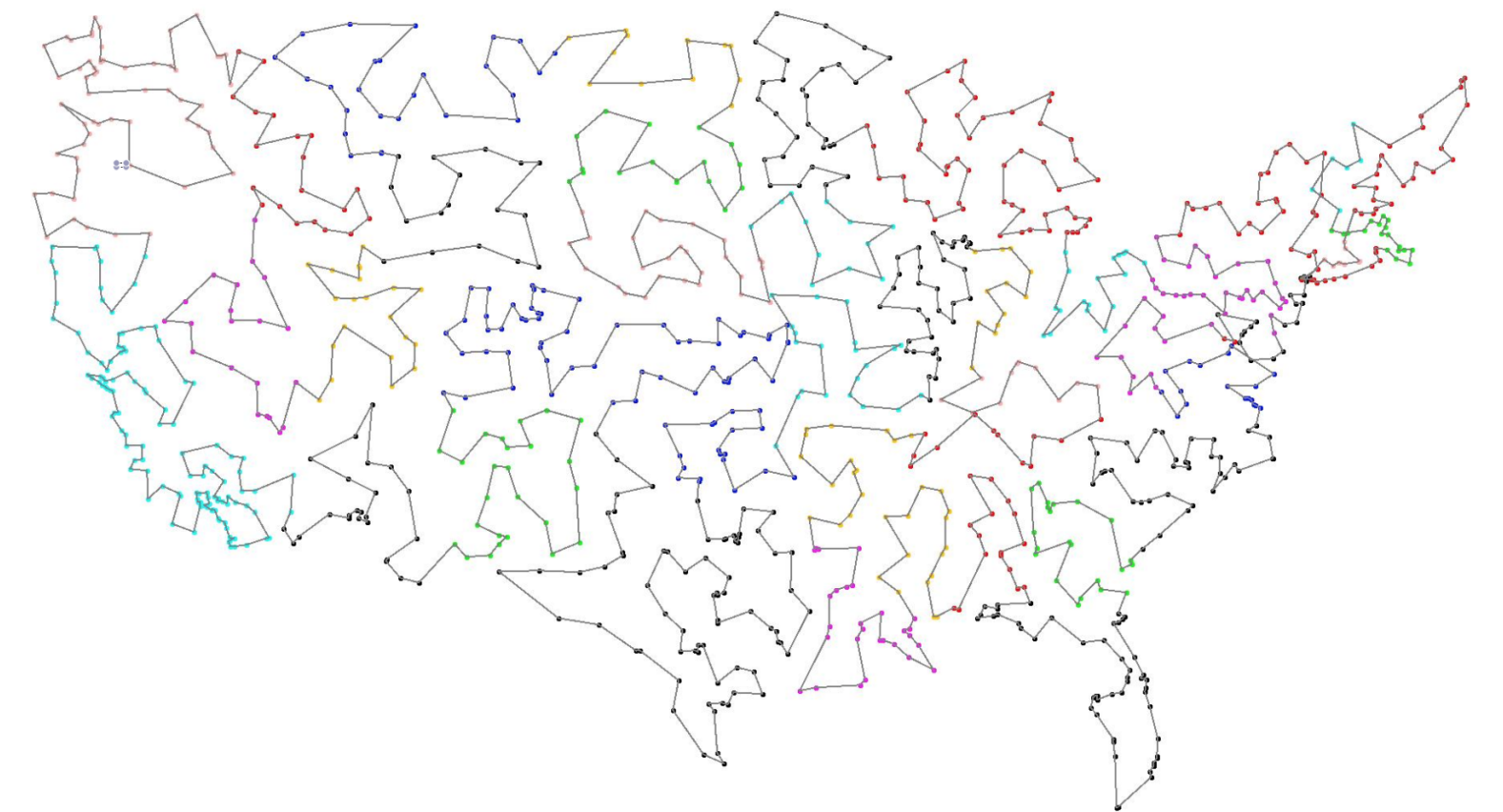
Goal: find a minimum-cost Hamiltonian tour over n cities



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)



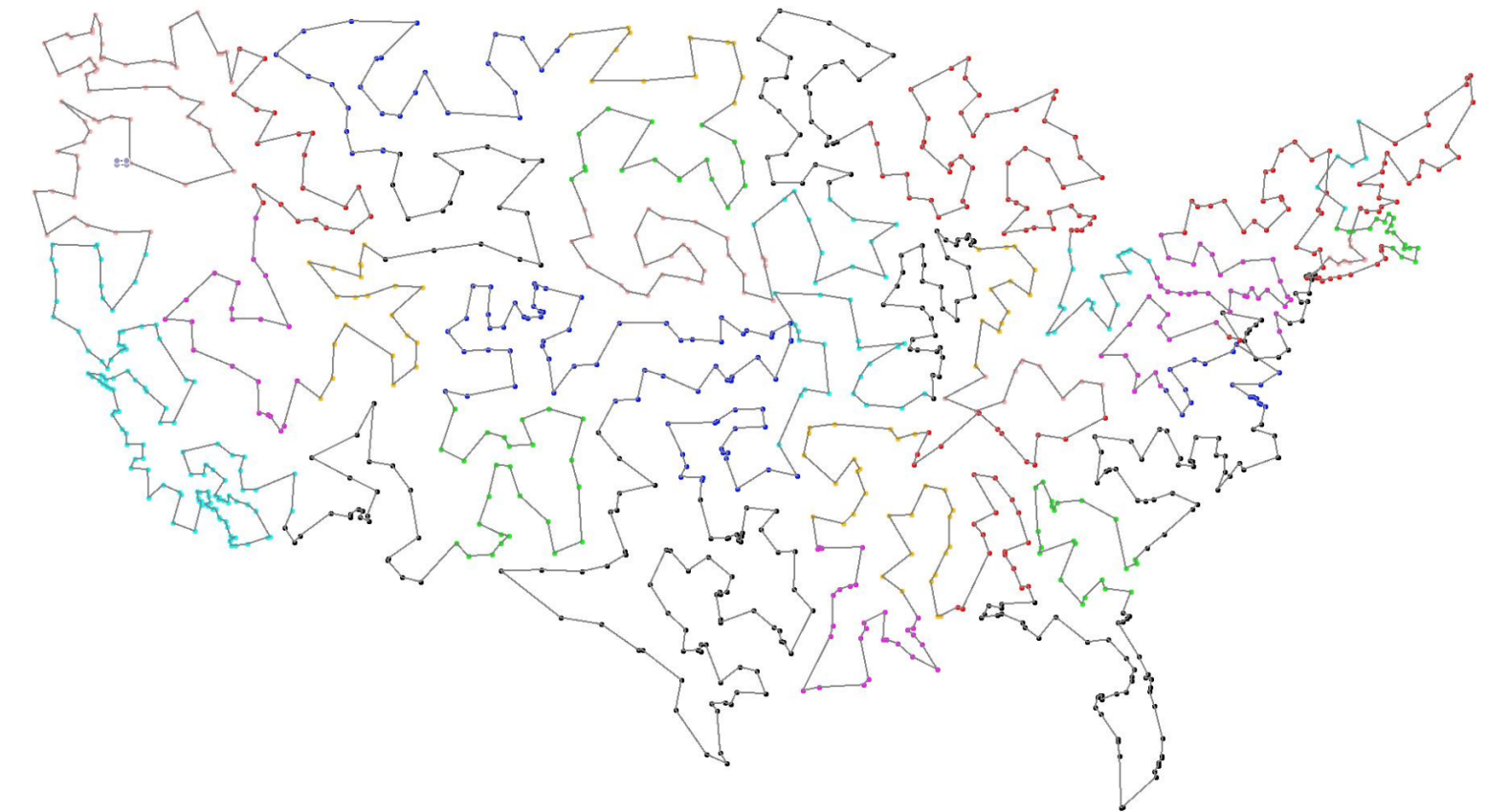
Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

Order variables: $u_i \in \{1, \dots, n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$



Example: Traveling salesman problem

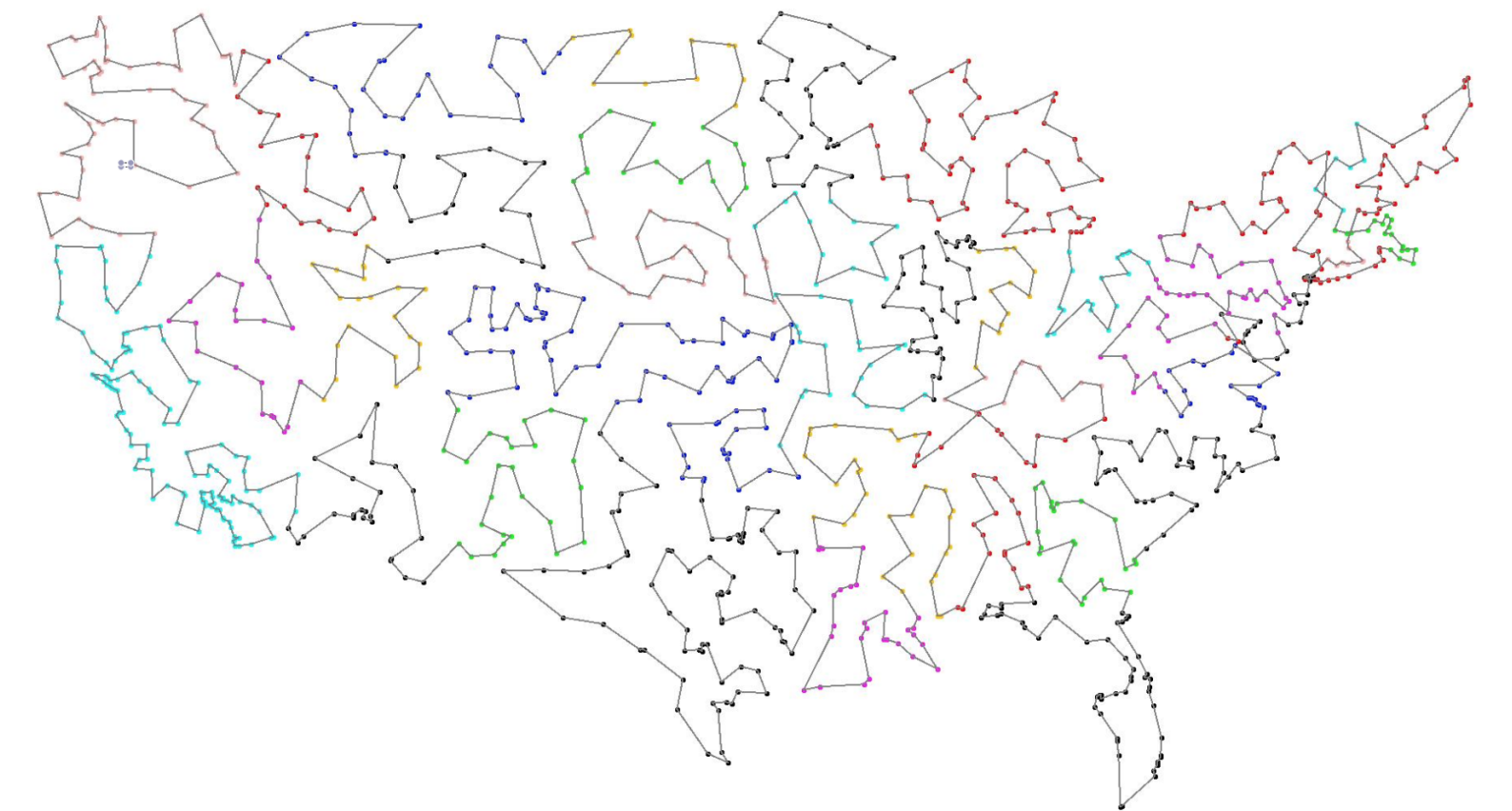
Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

Order variables: $u_i \in \{1, \dots, n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

Objective: $\min \sum_{i \neq j} c_{ij} x_{ij}$ (c_{ij} is the cost to travel $i \rightarrow j$)



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

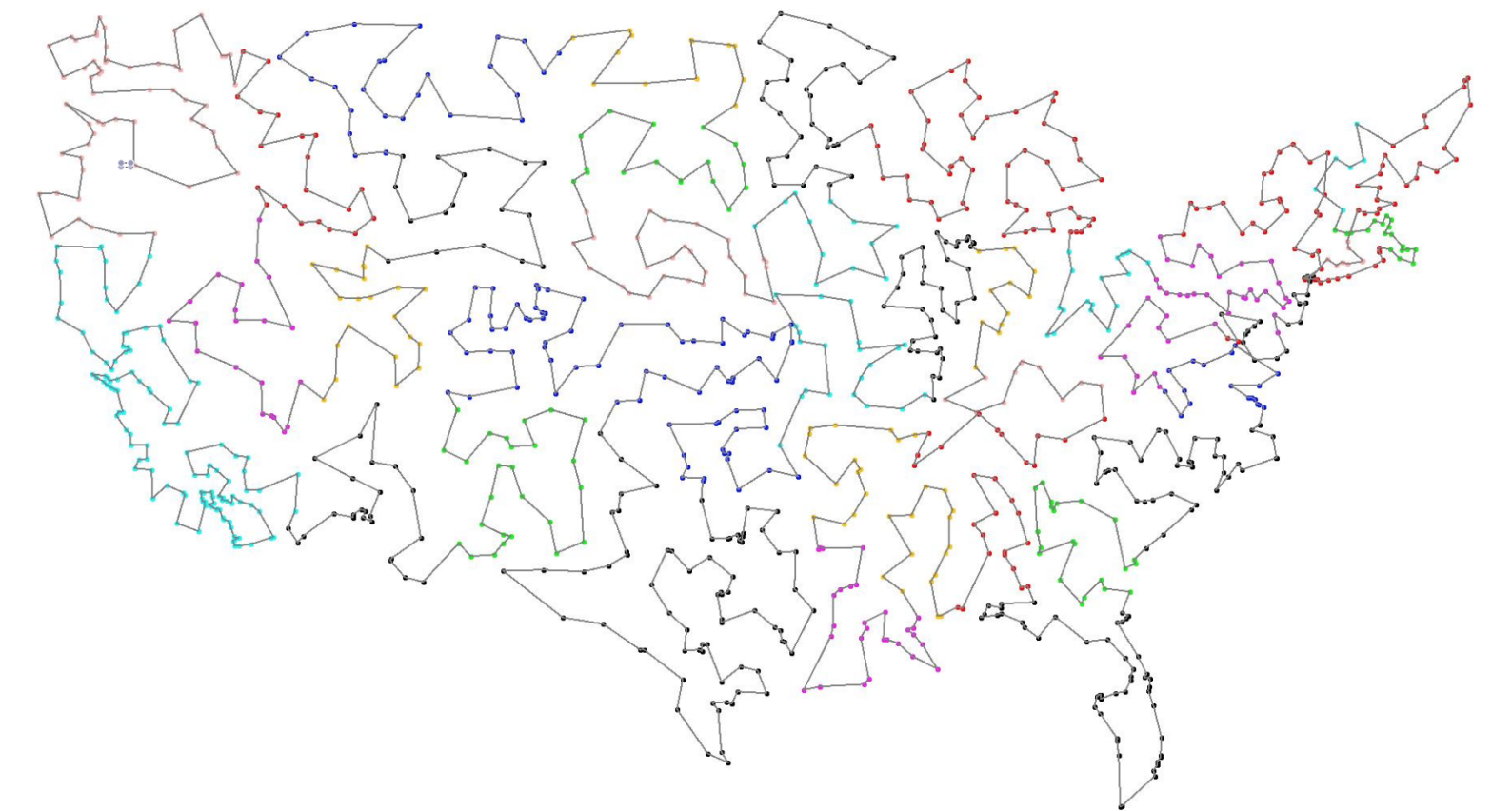
Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

Order variables: $u_i \in \{1, \dots, n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

Objective: $\min \sum_{i \neq j} c_{ij} x_{ij}$ (c_{ij} is the cost to travel $i \rightarrow j$)

Degree constraints (enter/leave exactly once): $\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1, \quad \forall i$



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

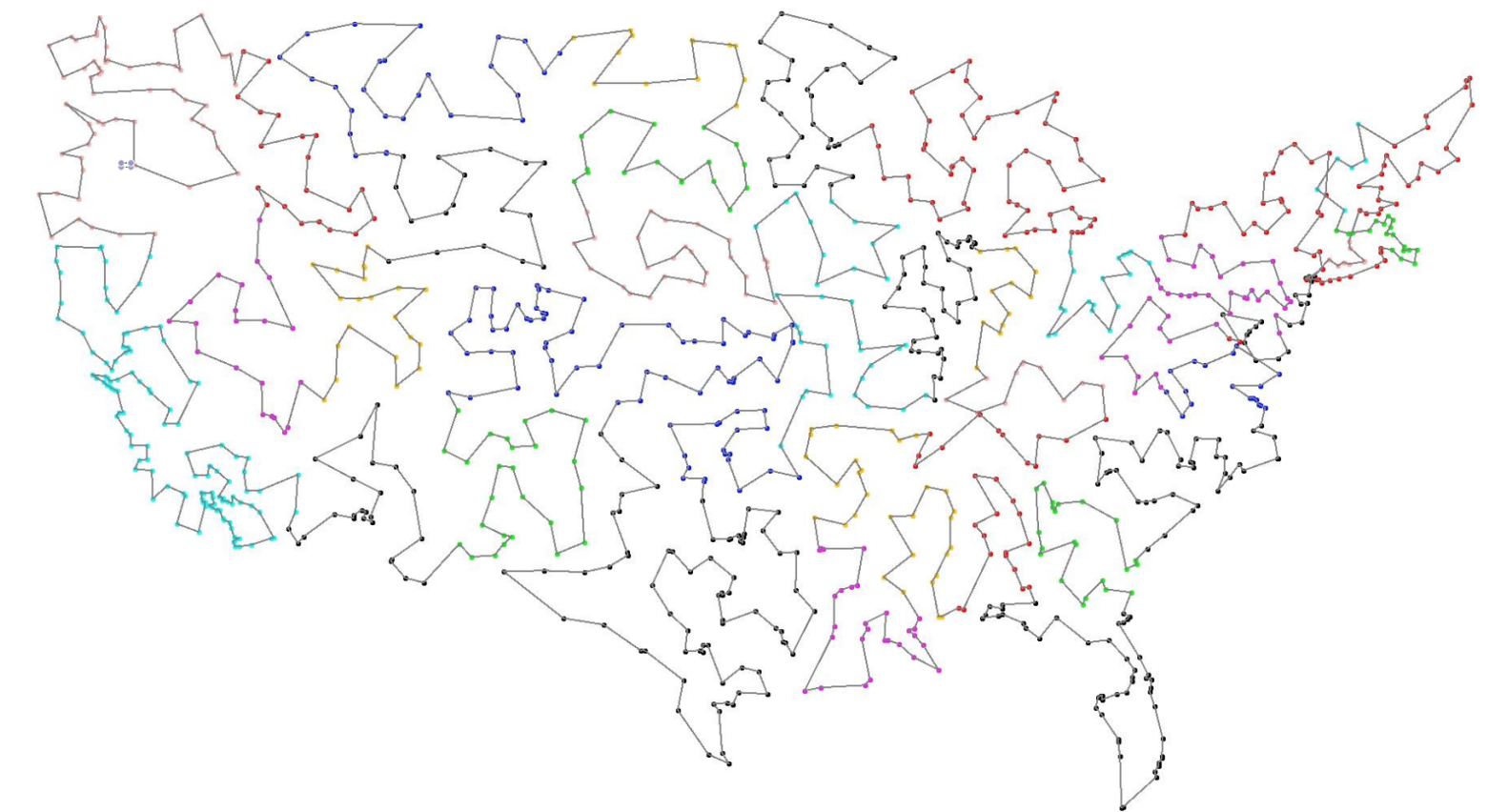
Order variables: $u_i \in \{1, \dots, n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

Objective: $\min \sum_{i \neq j} c_{ij} x_{ij}$ (c_{ij} is the cost to travel $i \rightarrow j$)

Degree constraints (enter/leave exactly once): $\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1, \quad \forall i$

Subtour-elimination: $u_i - u_j + nx_{ij} \leq n - 1, \quad \forall i \neq j, \quad i, j \in \{2, \dots, n\}$



Example: Traveling salesman problem

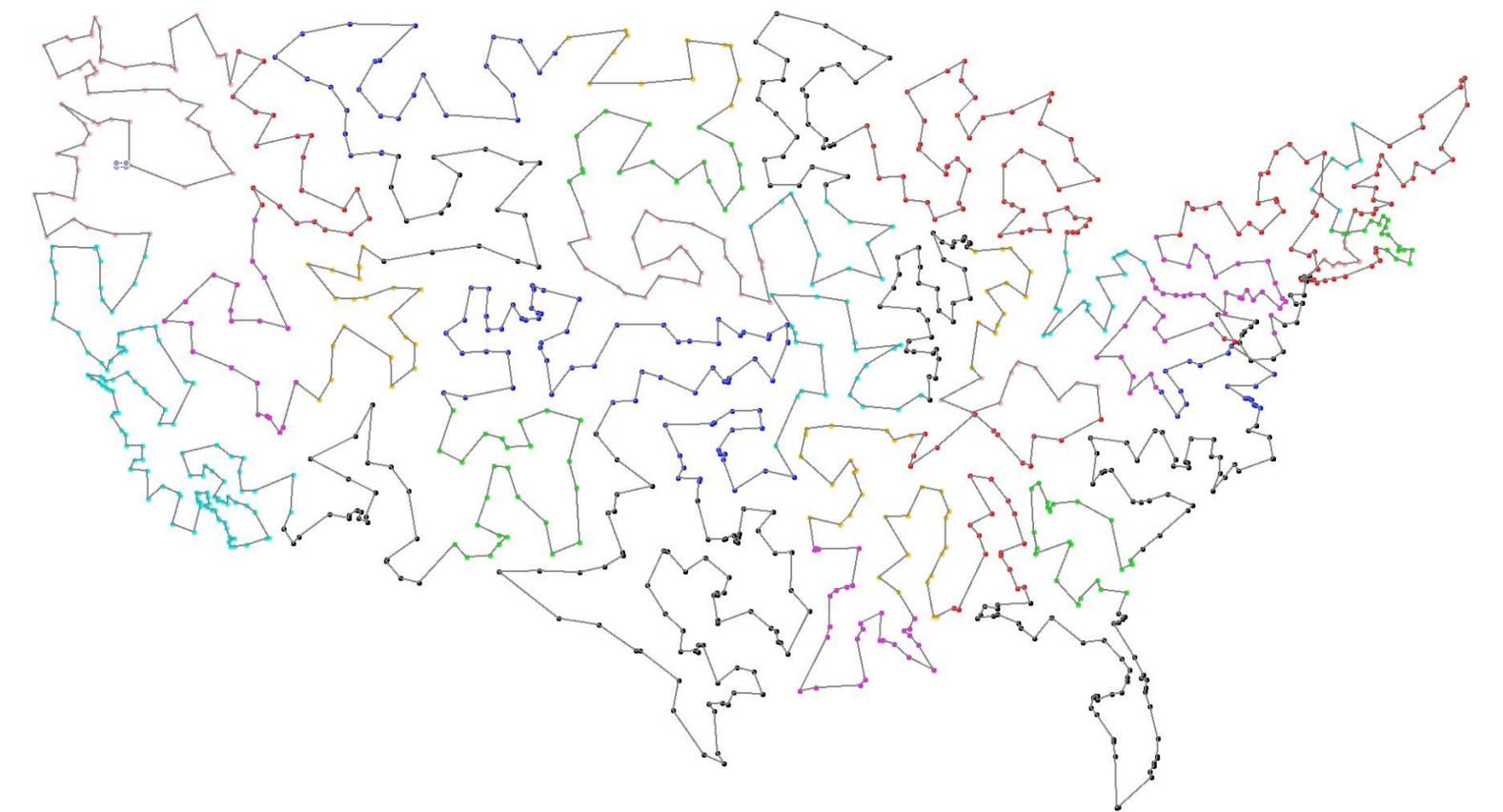
Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

Order variables: $u_i \in \{1,\dots,n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

(Very simple) example of a CP found by EvoCut:



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

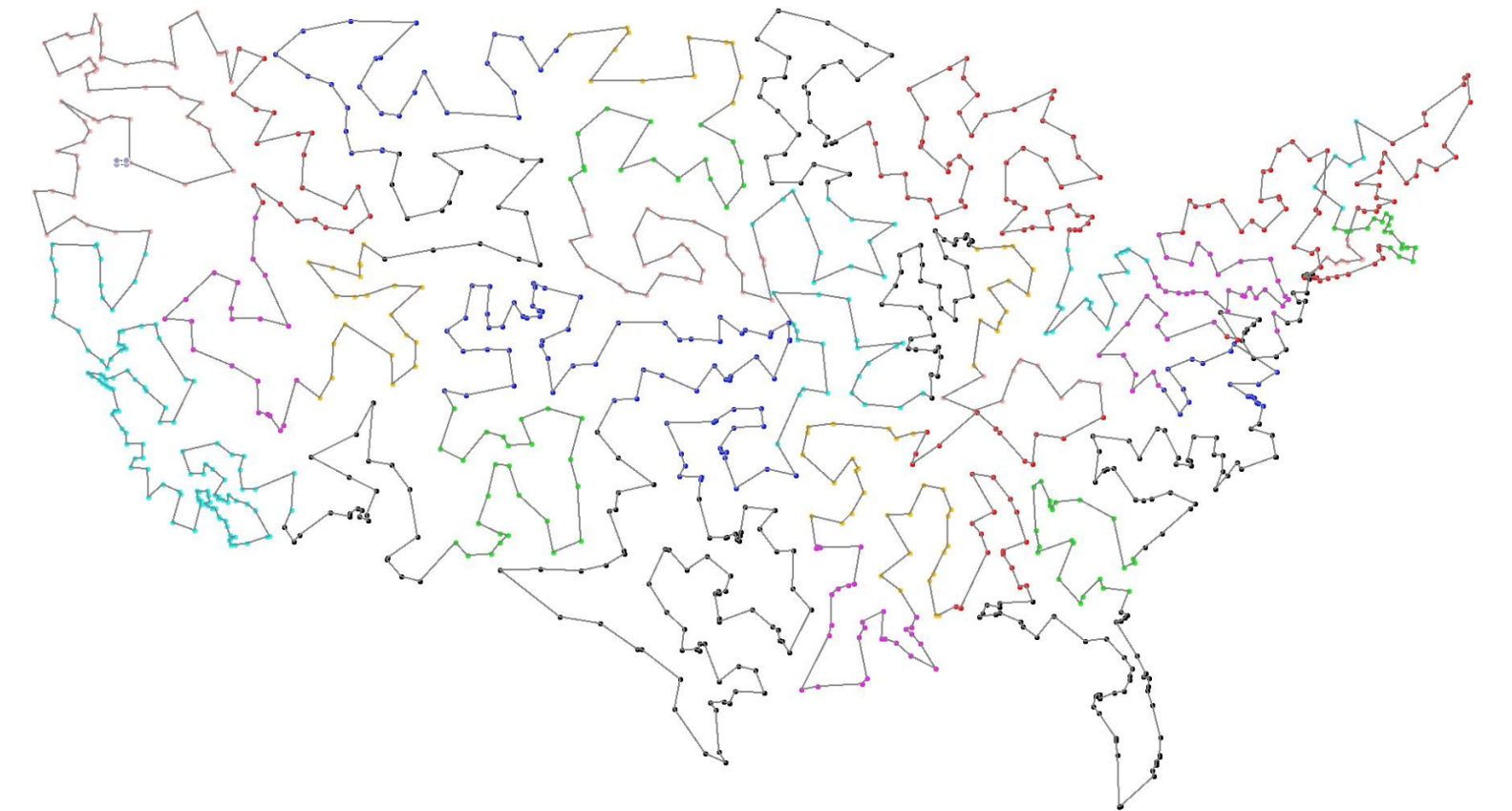
Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

Order variables: $u_i \in \{1, \dots, n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

(Very simple) example of a CP found by EvoCut:

$$u_j \leq 2 + (n - 2)(1 - x_{1j}), \forall j \in \{2, \dots, n\}$$



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

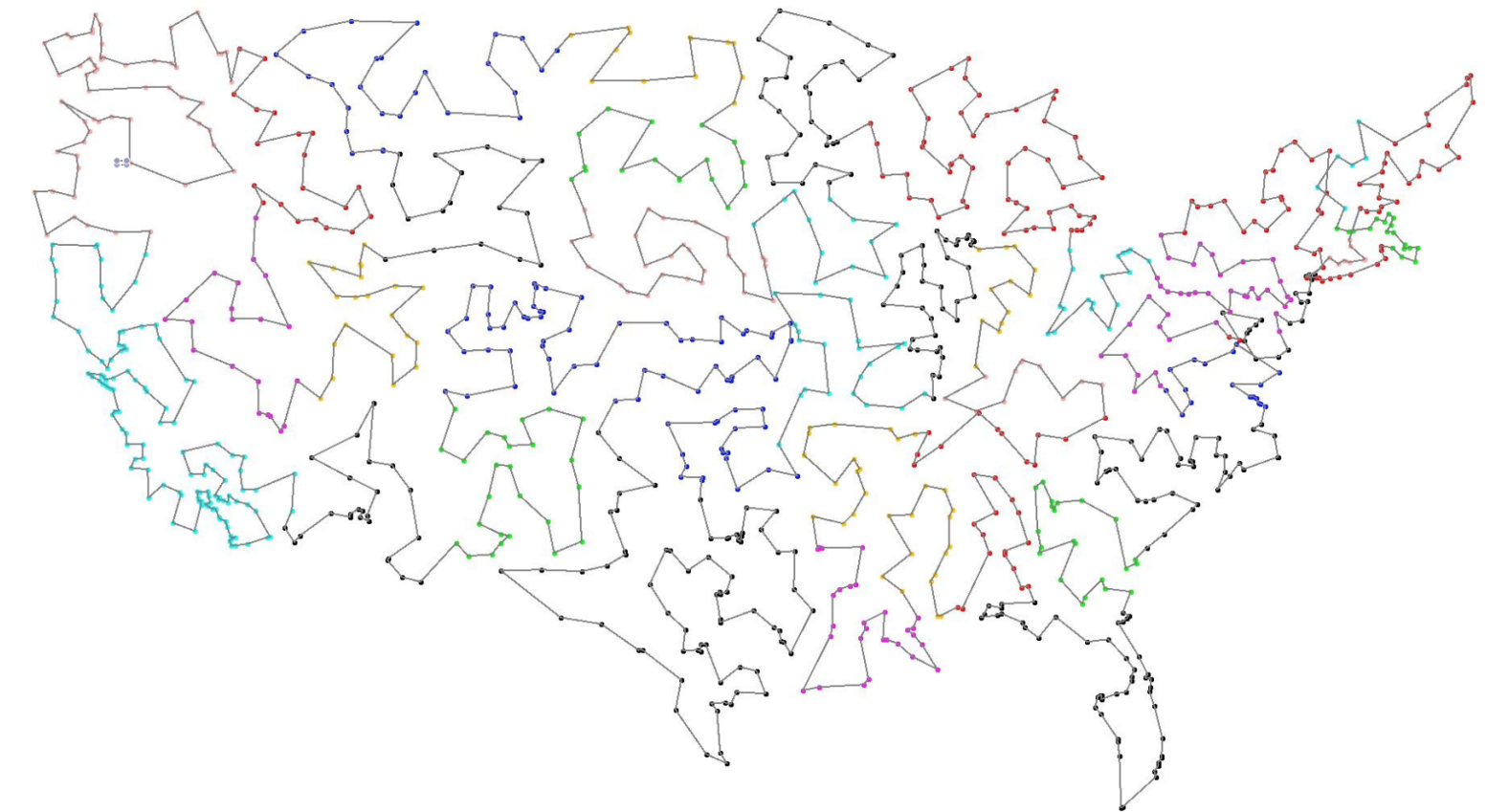
Order variables: $u_i \in \{1,\dots,n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

(Very simple) example of a CP found by EvoCut:

$$u_j \leq 2 + (n - 2)(1 - x_{1j}), \forall j \in \{2,\dots,n\}$$

- If $x_{1j} = 1$, $u_j \leq 2$ (travel $1 \rightarrow j$)



Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \rightarrow j$)

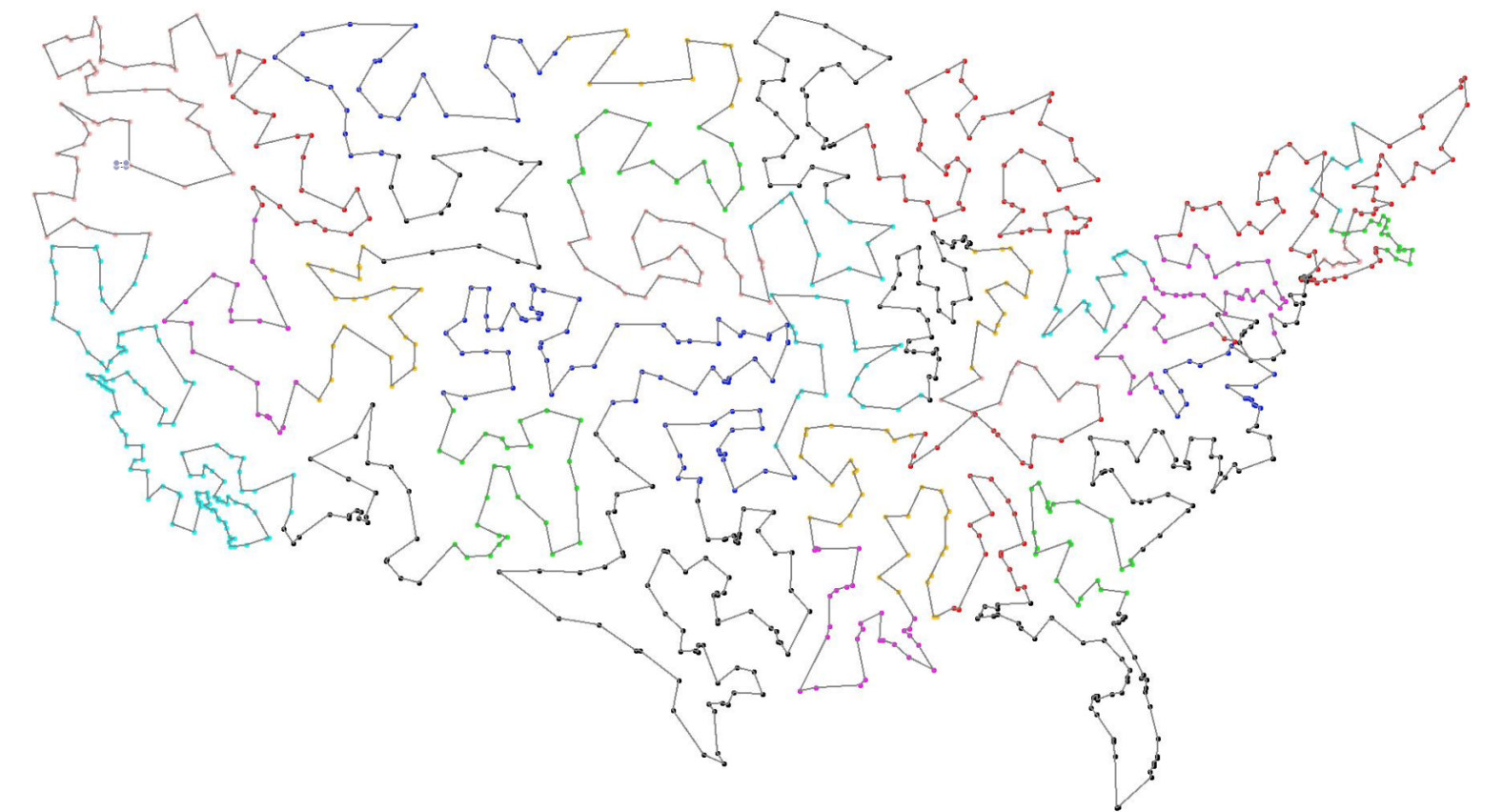
Order variables: $u_i \in \{1,\dots,n\}$ (visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \rightarrow j$

(Very simple) example of a CP found by EvoCut:

$$u_j \leq 2 + (n - 2)(1 - x_{1j}), \forall j \in \{2,\dots,n\}$$

- If $x_{1j} = 1$, $u_j \leq 2$ (travel $1 \rightarrow j$)
- If $x_{1j} = 0$, $u_j \leq n$



Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs

Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs
- On D_e : run baseline solver $\forall z \in D_e$ under fixed computational budget

Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs
- On D_e : run baseline solver $\forall z \in D_e$ under fixed computational budget
 - Record terminal **optimality gap** $\text{gap}_{\text{ref}}(z)$

Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs
- On D_e : run baseline solver $\forall z \in D_e$ under fixed computational budget
 - Record terminal **optimality gap** $\text{gap}_{\text{ref}}(z)$
- On D_v : run baseline solver to optimality and store **optimal solution**

Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs
- On D_e : run baseline solver $\forall z \in D_e$ under fixed computational budget
 - Record terminal **optimality gap** $\text{gap}_{\text{ref}}(z)$
- On D_v : run baseline solver to optimality and store **optimal solution**
 - Also store **solution to LP relaxation**

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry
 2. **OSP**: maintains feasibility of optimal solutions across D_v

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry
 2. **OSP**: maintains feasibility of optimal solutions across D_v
 3. **Usefulness**: separate LP-optimal solution on *some* D_v instance

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry
 2. **OSP**: maintains feasibility of optimal solutions across D_v
 3. **Usefulness**: separate LP-optimal solution on *some* D_v instance
- Run baseline solver $\forall z \in D_e$ instances under fixed computational budget

Method overview

How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry
 2. **OSP**: maintains feasibility of optimal solutions across D_v
 3. **Usefulness**: separate LP-optimal solution on *some* D_v instance
- Run baseline solver $\forall z \in D_e$ instances under fixed computational budget
 - Record terminal optimality gap $\text{gap}_{\text{cut}}(z)$ **with cut**

Method overview

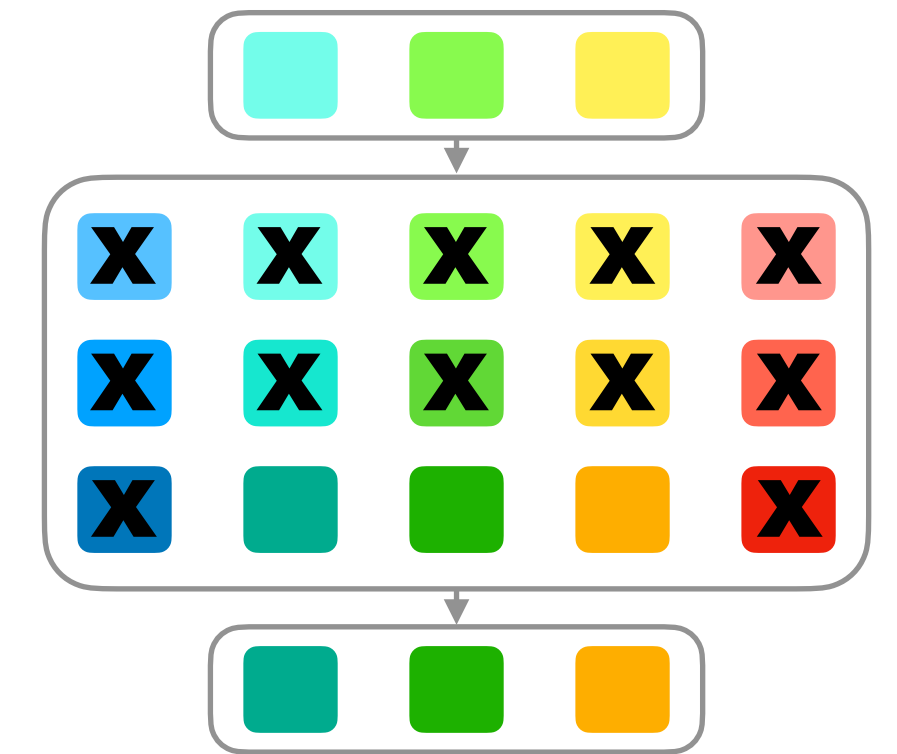
How to verify and evaluate a candidate cut

- Verification on D_v :
 1. Code must **compile**; errors trigger diagnostic prompt, retry
 2. **OSP**: maintains feasibility of optimal solutions across D_v
 3. **Usefulness**: separate LP-optimal solution on *some* D_v instance
- Run baseline solver $\forall z \in D_e$ instances under fixed computational budget
 - Record terminal optimality gap $\text{gap}_{\text{cut}}(z)$ **with cut**
- **Fitness**: average relative gap change over D_e :
$$\frac{\text{gap}_{\text{ref}}(z) - \text{gap}_{\text{cut}}(z)}{\text{gap}_{\text{ref}}(z)}$$

Method overview

Candidate cut generation

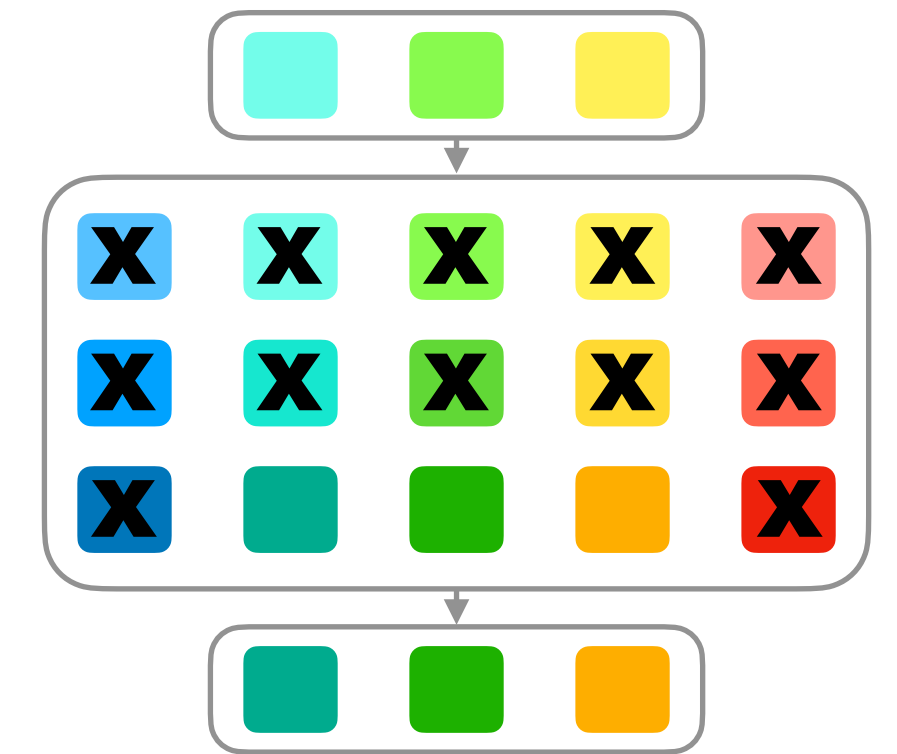
1. **Initializer LLM** seeds candidate cuts



Method overview

Candidate cut generation

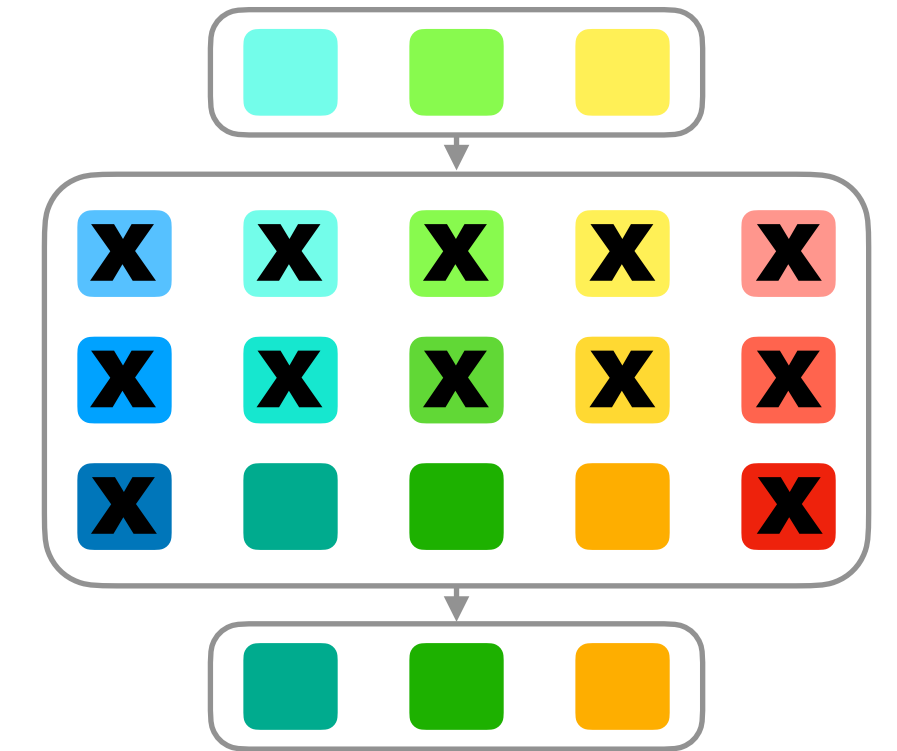
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description



Method overview

Candidate cut generation

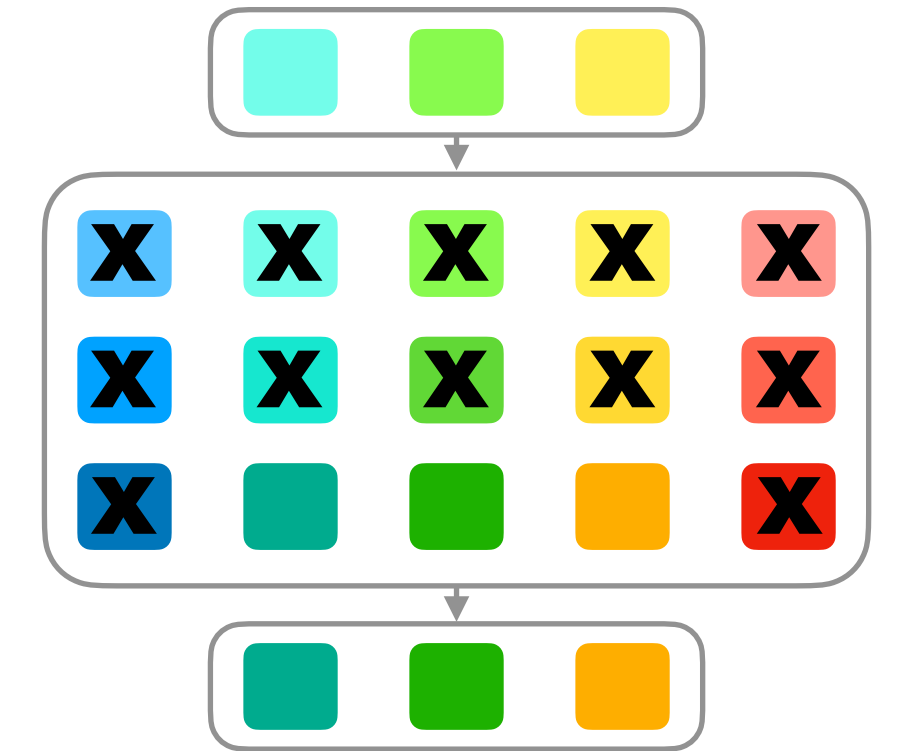
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations



Method overview

Candidate cut generation

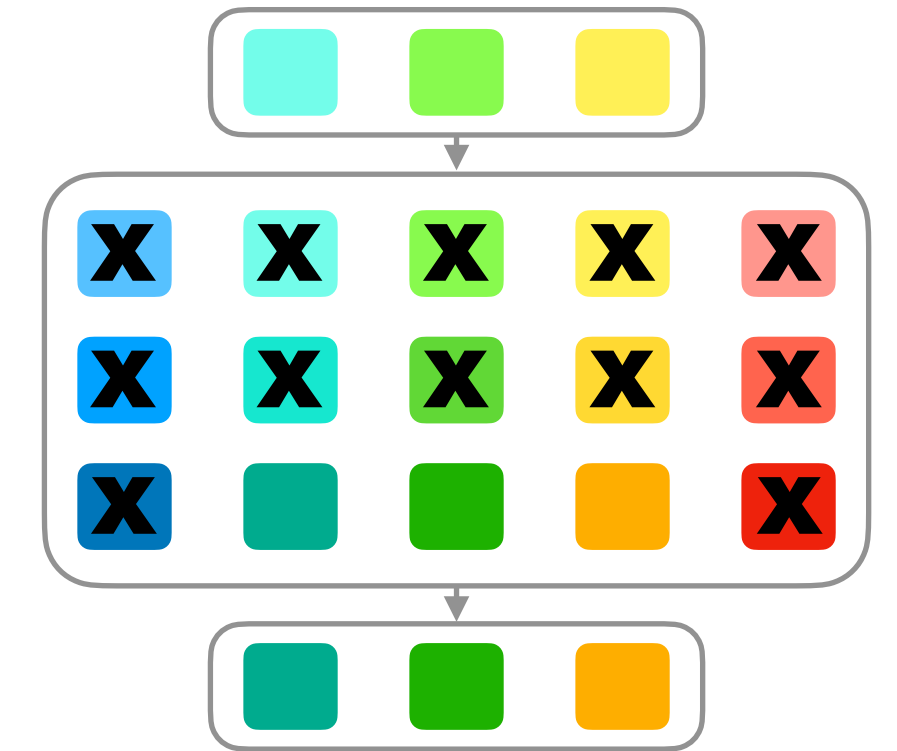
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations
 - **Elitism**: carry top cuts into next generation



Method overview

Candidate cut generation

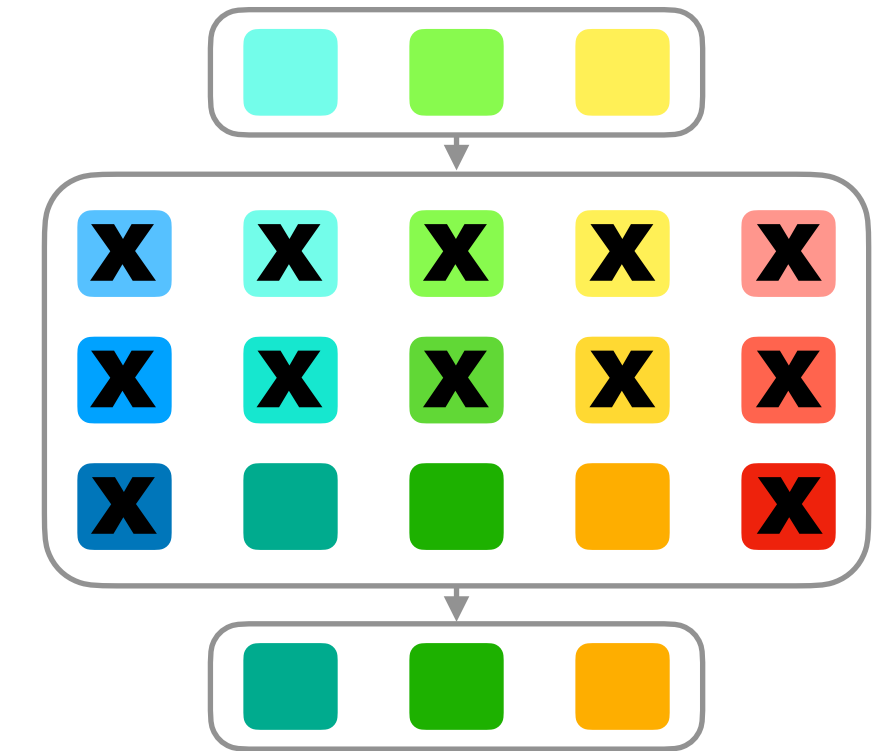
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations
 - **Elitism**: carry top cuts into next generation
 - **Selection**: pick parents with probability proportional to fitness



Method overview

Candidate cut generation

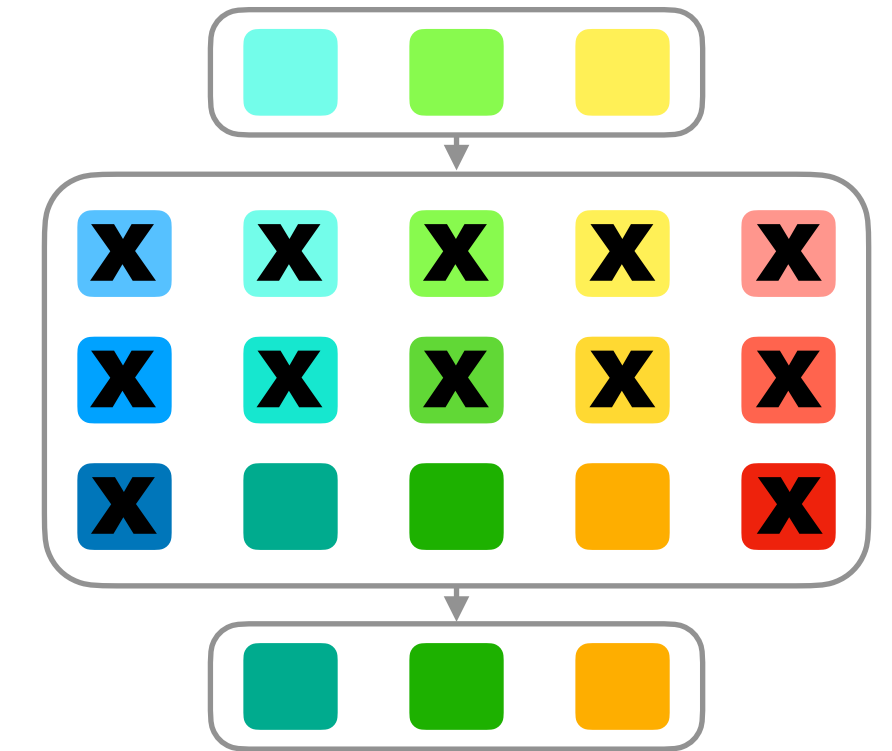
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations
 - **Elitism**: carry top cuts into next generation
 - **Selection**: pick parents with probability proportional to fitness
 - **Reproduce**: crossover (merge two parents) or mutation



Method overview

Candidate cut generation

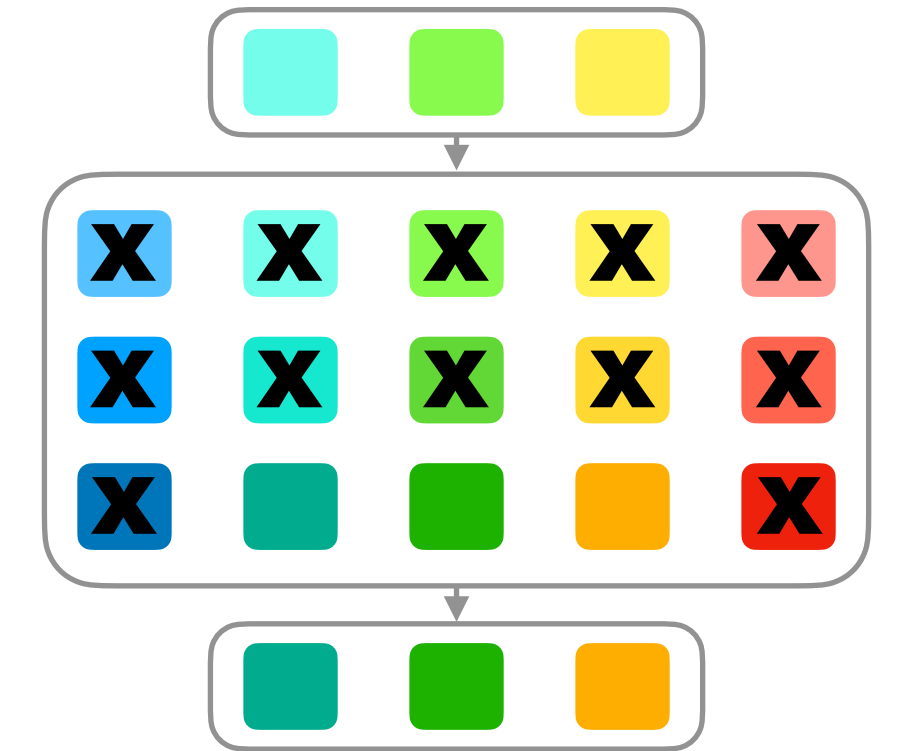
1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations
 - **Elitism**: carry top cuts into next generation
 - **Selection**: pick parents with probability proportional to fitness
 - **Reproduce**: crossover (merge two parents) or mutation
3. **Verify and evaluate** offspring before including in population



Method overview

Candidate cut generation

1. **Initializer LLM** seeds candidate cuts
 - Uses model code and natural language description
2. Population **evolved** for T generations
 - **Elitism**: carry top cuts into next generation
 - **Selection**: pick parents with probability proportional to fitness
 - **Reproduce**: crossover (merge two parents) or mutation
3. **Verify and evaluate** offspring before including in population
 - Failed offspring triggers feedback; retry up to max



Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP
 - Multi-Commodity Network Design (MCND)

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP
 - Multi-Commodity Network Design (MCND)
 - Capacitated Warehouse Location Problem (CWLP)

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP
 - Multi-Commodity Network Design (MCND)
 - Capacitated Warehouse Location Problem (CWLP)
 - Job Shop Scheduling Problem (JSSP)

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP
 - Multi-Commodity Network Design (MCND)
 - Capacitated Warehouse Location Problem (CWLP)
 - Job Shop Scheduling Problem (JSSP)
- $|D_e| = 10, |D_v| = 2$; drawn from synthetic generators

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
 - TSP
 - Multi-Commodity Network Design (MCND)
 - Capacitated Warehouse Location Problem (CWLP)
 - Job Shop Scheduling Problem (JSSP)
- $|D_e| = 10, |D_v| = 2$; drawn from synthetic generators
- **Test set** D_t : public datasets; 40 hard medium/large instances each

Snapshot of results

Checkpoints	5 s	10 s	50 s	150 s	300 s	OSP rate (%)
TSP	16.3 ± 24.9	15.4 ± 27.3	27.7 ± 31.1	44.4 ± 27.7	57.4 ± 26.3	100
MCND	9.4 ± 21.1	6.3 ± 22.0	11.7 ± 19.1	10.4 ± 18.4	17.1 ± 20.2	100
CWLP	-6.9 ± 17.0	-8.3 ± 15.1	24.0 ± 24.9	42.5 ± 21.3	46.2 ± 41.1	100
JSSP	22.8 ± 18.3	28.8 ± 19.7	39.1 ± 22.8	34.5 ± 22.1	37.3 ± 22.0	100

Relative mean gap improvement over D_t

Snapshot of results

Checkpoints	5 s	10 s	50 s	150 s	300 s	OSP rate (%)
TSP	16.3 ± 24.9	15.4 ± 27.3	27.7 ± 31.1	44.4 ± 27.7	57.4 ± 26.3	100
MCND	9.4 ± 21.1	6.3 ± 22.0	11.7 ± 19.1	10.4 ± 18.4	17.1 ± 20.2	100
CWLP	-6.9 ± 17.0	-8.3 ± 15.1	24.0 ± 24.9	42.5 ± 21.3	46.2 ± 41.1	100
JSSP	22.8 ± 18.3	28.8 ± 19.7	39.1 ± 22.8	34.5 ± 22.1	37.3 ± 22.0	100

Relative mean gap improvement over D_t

- **Evolution helps:** initializer-only cuts have lower success

Check (i) code rejections, (ii) OSP rejections, (iii) usefulness rejections, and (iv) fitness

Snapshot of results

Checkpoints	5 s	10 s	50 s	150 s	300 s	OSP rate (%)
TSP	16.3 ± 24.9	15.4 ± 27.3	27.7 ± 31.1	44.4 ± 27.7	57.4 ± 26.3	100
MCND	9.4 ± 21.1	6.3 ± 22.0	11.7 ± 19.1	10.4 ± 18.4	17.1 ± 20.2	100
CWLP	-6.9 ± 17.0	-8.3 ± 15.1	24.0 ± 24.9	42.5 ± 21.3	46.2 ± 41.1	100
JSSP	22.8 ± 18.3	28.8 ± 19.7	39.1 ± 22.8	34.5 ± 22.1	37.3 ± 22.0	100

Relative mean gap improvement over D_t

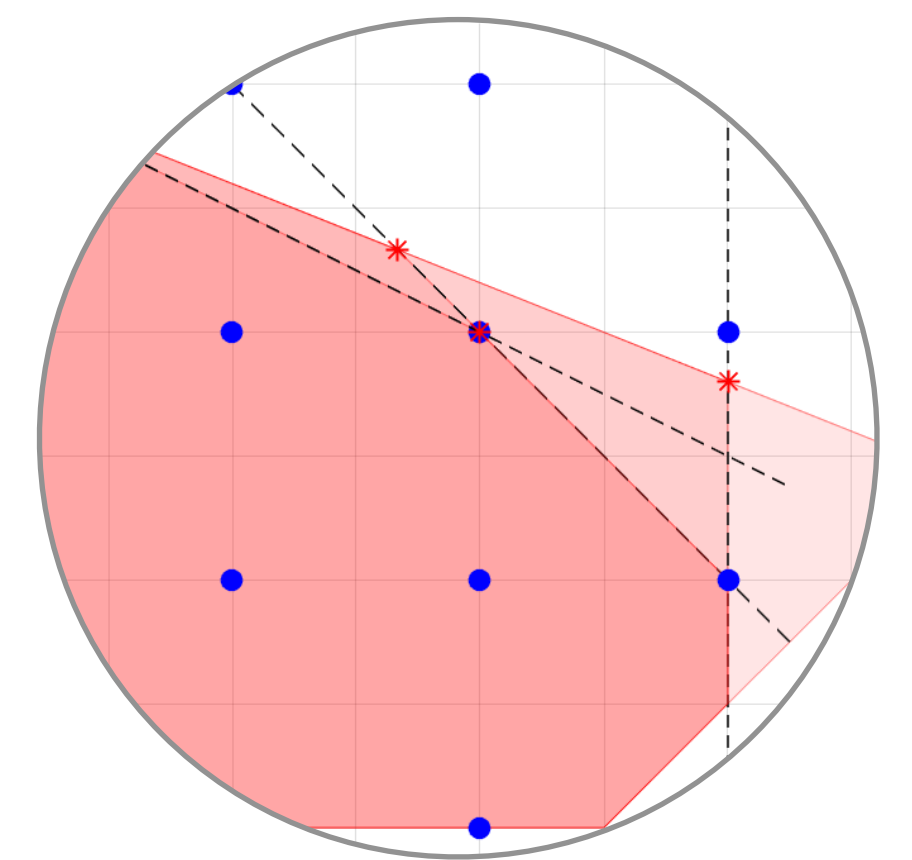
- **Evolution helps:** initializer-only cuts have lower success

Check (i) code rejections, (ii) OSP rejections, (iii) usefulness rejections, and (iv) fitness

- Mutation/crossover succeed 63–82% attempts; initializer 25.4%

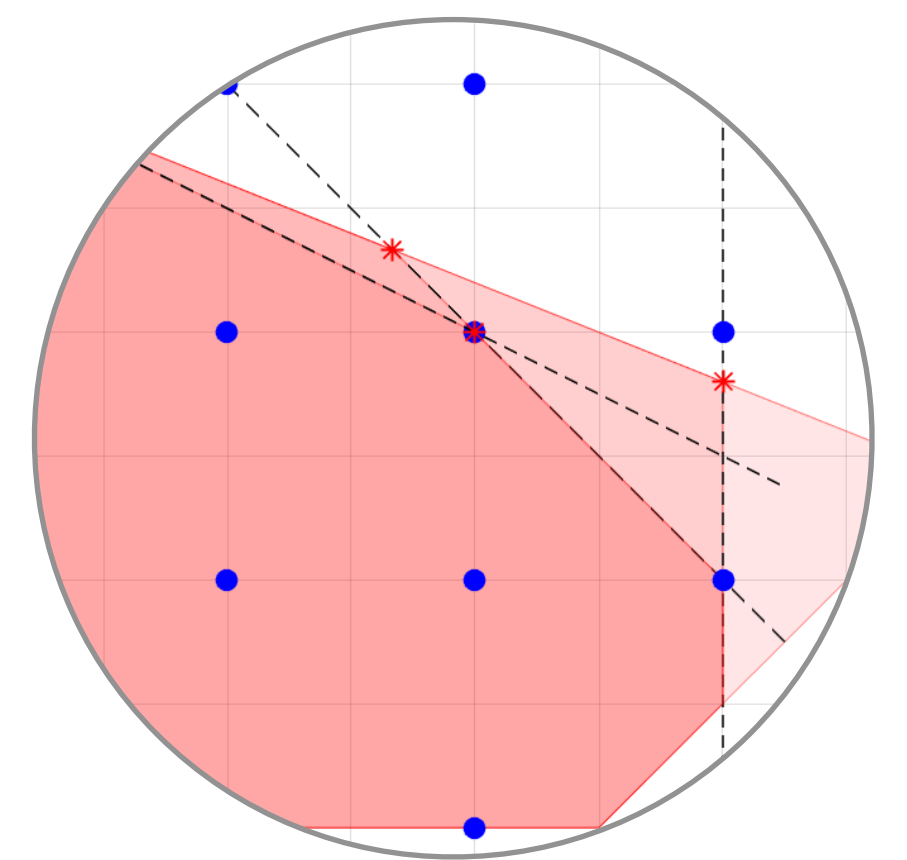
Takeaways

- **EvoCut**: Evolutionary search automates cut discovery



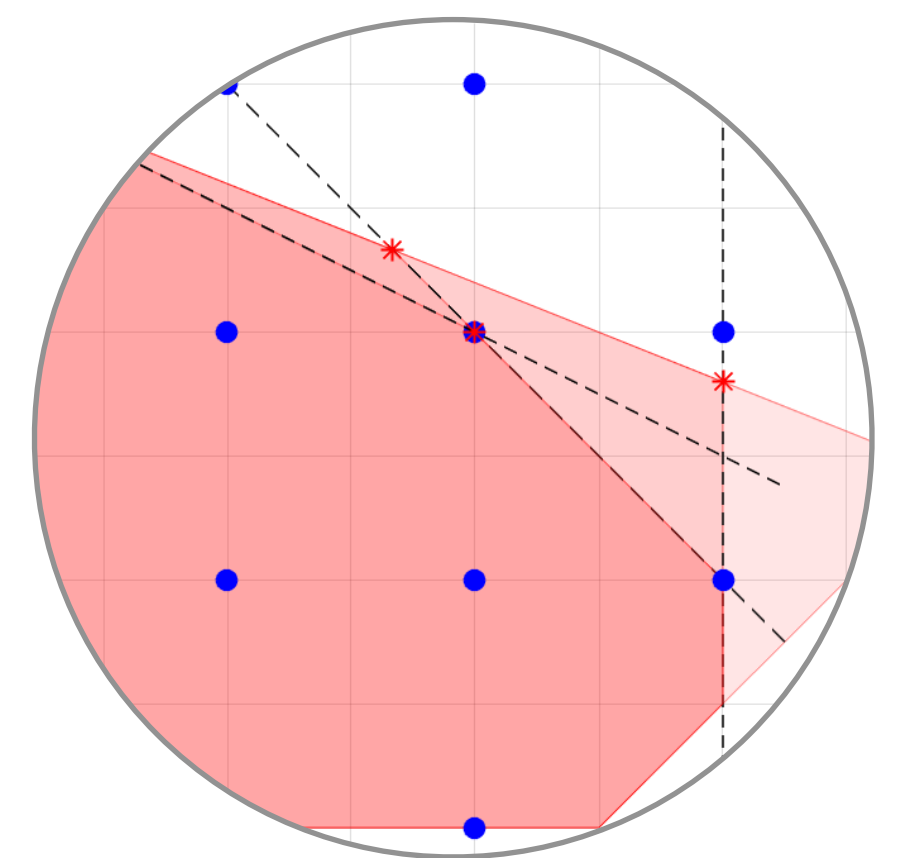
Takeaways

- **EvoCut**: Evolutionary search automates cut discovery
- Optimal-solution preservation and fractional separation verified **empirically**



Takeaways

- **EvoCut**: Evolutionary search automates cut discovery
- Optimal-solution preservation and fractional separation verified **empirically**
- Cuts improve **optimality-gap reduction** throughout solve trajectory



Outline

1. Background
2. LLMs for speeding up solvers
 - i. Solver configuration
 - ii. Evolutionary search for MILP heuristics
 - a. Background
 - b. Cut selection
 - c. Large neighborhood search**
 - d. Diving heuristics
3. Takeaways

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance
 - Without ML: Fischetti, Lodi, MP'03; Danna et al., MP'05; ...

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance
 - Without ML: Fischetti, Lodi, MP'03; Danna et al., MP'05; ...
 - With ML: Song et al., NeurIPS'20; Wu et al., NeurIPS'21; Huang et al., ICML'23; ...

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance
 - Without ML: Fischetti, Lodi, MP'03; Danna et al., MP'05; ...
 - With ML: Song et al., NeurIPS'20; Wu et al., NeurIPS'21; Huang et al., ICML'23; ...
- Prior neighborhood selection methods need **expertise** or **heavy training**

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance
 - Without ML: Fischetti, Lodi, MP'03; Danna et al., MP'05; ...
 - With ML: Song et al., NeurIPS'20; Wu et al., NeurIPS'21; Huang et al., ICML'23; ...
- Prior neighborhood selection methods need **expertise** or **heavy training**
- Propose **LLM-LNS**: LLM-guided neighborhood scoring for MILPs

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem
- Neighborhood choice controls search power and runtime tradeoff

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem
- Neighborhood choice controls search power and runtime tradeoff
 - **Small neighborhoods**: fast but limited improvement

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem
- Neighborhood choice controls search power and runtime tradeoff
 - **Small neighborhoods**: fast but limited improvement
 - **Large neighborhoods**: powerful but expensive

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

For each integer variable x_i :

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

For each integer variable x_i :

- Fix all other integer variables to current values

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

For each integer variable x_i :

- Fix all other integer variables to current values
- Relax x_i ; solve resulting LP

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

For each integer variable x_i :

- Fix all other integer variables to current values
- Relax x_i ; solve resulting LP
- Measure deviation from integer solution value

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
 - **Destroy**: fix most variables; free a selected subset
 - **Repair**: solve resulting MILP subproblem

For each integer variable x_i :

- Fix all other integer variables to current values
- Relax x_i ; solve resulting LP
- Measure deviation from integer solution value
- Free/destroy variables with largest deviation

LLM-LNS framework: Method overview

Inner-outer agent

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**
 - Given feasible MILP solution, plus bounds and objective coefficients for variables

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS
- You must score all variables for neighborhood selection decisions

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS
- You must score all variables for neighborhood selection decisions
- Add some randomness in subset choice to avoid local optima and stagnation

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**
 - Given feasible MILP solution, plus bounds and objective coefficients for variables
 - Goal: improve the current solution using LNS
 - Description of LNS
 - You must score all variables for neighborhood selection decisions
 - Add some randomness in subset choice to avoid local optima and stagnation
 - m prior scoring rules: description, average fitness (objective value) over training set, code

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS
- You must score all variables for neighborhood selection decisions
- Add some randomness in subset choice to avoid local optima and stagnation
- m prior scoring rules: description, average fitness (objective value) over training set, code
- Create a totally different new scoring rule

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
 - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS
- You must score all variables for neighborhood selection decisions
- Add some randomness in subset choice to avoid local optima and stagnation
- m prior scoring rules: description, average fitness (objective value) over training set, code
- Create a totally different new scoring rule

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
 - **Example prompt:**

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
 - **Example prompt:**

- Goal: solve minimization problem by using LLM-generated LNS heuristics

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
 - **Example prompt:**
 - Goal: solve minimization problem by using LLM-generated LNS heuristics
 - We already designed several initial prompts and collected the LLM's resulting heuristics

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
 - **Example prompt:**

- Goal: solve minimization problem by using LLM-generated LNS heuristics
- We already designed several initial prompts and collected the LLM's resulting heuristics
- Analyze each prompt and the objective function values achieved by LNS heuristic

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
 - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
 - **Example prompt:**

- Goal: solve minimization problem by using LLM-generated LNS heuristics
- We already designed several initial prompts and collected the LLM's resulting heuristics
- Analyze each prompt and the objective function values achieved by LNS heuristic
- Create new prompt w/ different structure, motivated by insights from prompt-value pairs

Experimental setup

Benchmarks

- **Benchmarks:**
 - Set cover (SC)
 - Min vertex cover (MVC)
 - Max independent set (MIS)
 - See paper for mixed integer knapsack set (MIKS)

Experimental setup

Benchmarks

- **Benchmarks:**
 - Set cover (SC)
 - Min vertex cover (MVC)
 - Max independent set (MIS)
 - See paper for mixed integer knapsack set (MIKS)
- Tested on instances with (on the order of) 10^6 variables/constraints

Experimental setup

Benchmarks

- **Benchmarks:**
 - Set cover (SC)
 - Min vertex cover (MVC)
 - Max independent set (MIS)
 - See paper for mixed integer knapsack set (MIKS)
- Tested on instances with (on the order of) 10^6 variables/constraints
- Train on **small-scale instances** for scalable heuristic evolution

Experimental setup

Benchmarks

- **Benchmarks:**
 - Set cover (SC)
 - Min vertex cover (MVC)
 - Max independent set (MIS)
 - See paper for mixed integer knapsack set (MIKS)
- Tested on instances with (on the order of) 10^6 variables/constraints
- Train on **small-scale instances** for scalable heuristic evolution
 - Training instances are roughly **100x smaller** than the test instances

Experimental setup

Baselines

- **Random-LNS:** random variable neighborhoods in LNS

Experimental setup

Baselines

- **Random-LNS**: random variable neighborhoods in LNS
- **ACP**: adaptive constraint propagation guides neighborhood choice

Ye et al., AAAI'23

Experimental setup

Baselines

- **Random-LNS**: random variable neighborhoods in LNS
- **ACP**: adaptive constraint propagation guides neighborhood choice
Ye et al., AAAI'23
- **CL-LNS**: contrastive learning selects LNS neighborhoods
Huang et al., ICML'23

Experimental setup

Baselines

- **Random-LNS**: random variable neighborhoods in LNS
- **ACP**: adaptive constraint propagation guides neighborhood choice
Ye et al., AAAI'23
- **CL-LNS**: contrastive learning selects LNS neighborhoods
Huang et al., ICML'23
- **GNN&GBDT**: ML pipeline combining graphs and boosted trees
Ye et al., ICML'23

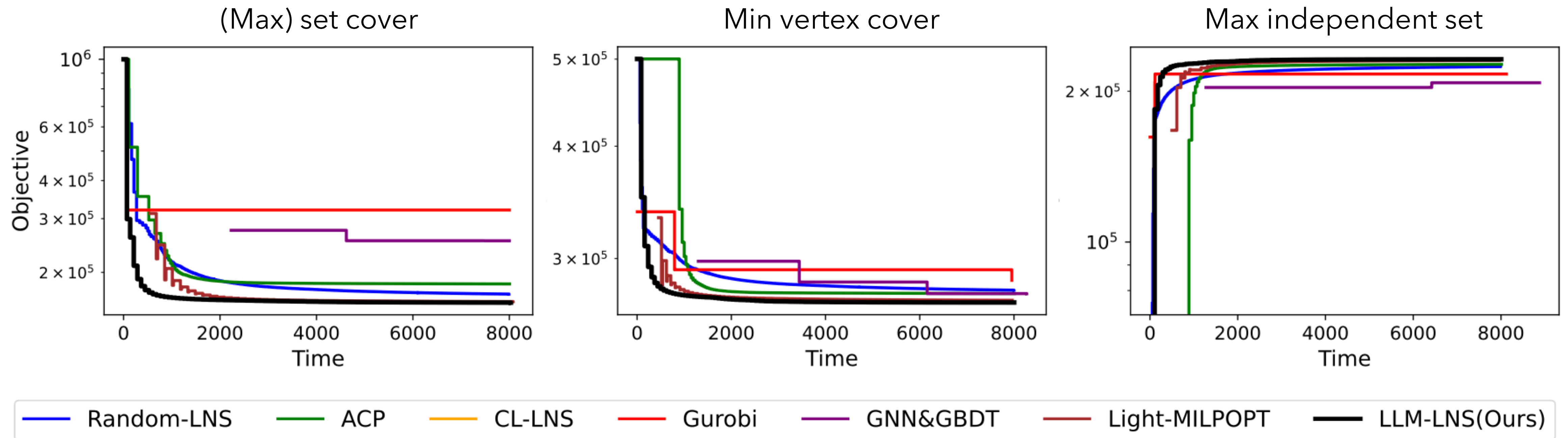
Experimental setup

Baselines

- **Random-LNS**: random variable neighborhoods in LNS
- **ACP**: adaptive constraint propagation guides neighborhood choice
Ye et al., AAAI'23
- **CL-LNS**: contrastive learning selects LNS neighborhoods
Huang et al., ICML'23
- **GNN&GBDT**: ML pipeline combining graphs and boosted trees
Ye et al., ICML'23
- **Light-MILPopt**: lightweight ML framework for MILP optimization
Ye et al., ICLR'23

Experiments

Snapshot of results



- LLM-LNS is mostly **Pareto optimal** (see paper for many more experiments)

Takeaways

- Prior neighborhood selection methods need **expertise** or **heavy training**

Takeaways

- Prior neighborhood selection methods need **expertise** or **heavy training**
- **LLM-LNS**: LLM-guided neighborhood scoring for MILPs

Takeaways

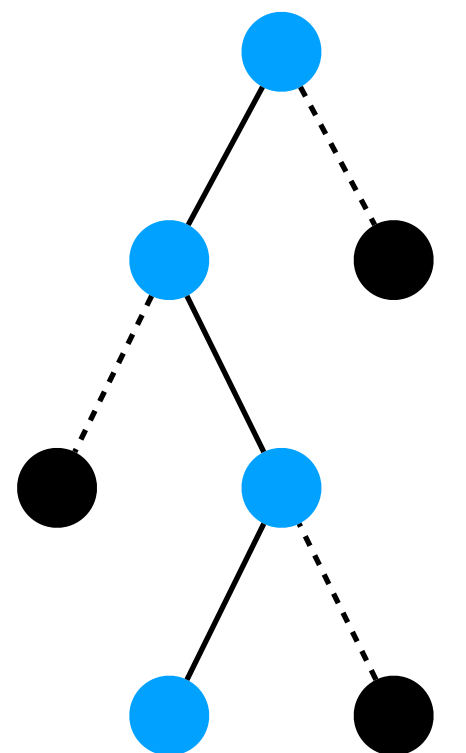
- Prior neighborhood selection methods need **expertise** or **heavy training**
- **LLM-LNS**: LLM-guided neighborhood scoring for MILPs
- Inner-outer agent evolves neighborhood selection heuristics **and** prompts

Outline

1. Background
2. LLMs for speeding up solvers
 - i. Solver configuration
 - ii. Evolutionary search for MILP heuristics
 - a. Background
 - b. Cut selection
 - c. Large neighborhood search
 - d. Diving heuristics**
3. Takeaways

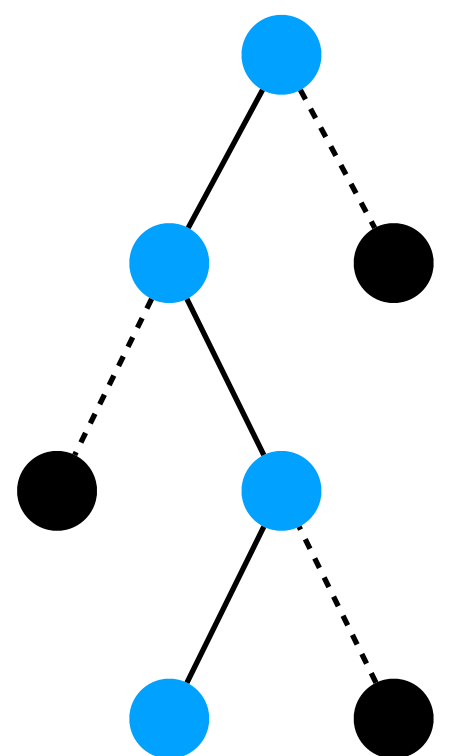
Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree



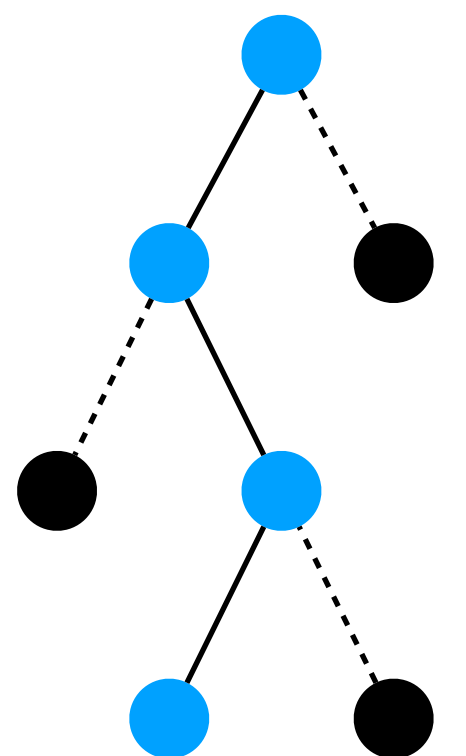
Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
 - Iteratively fixing fractional variables to integer values, solving LP relaxation



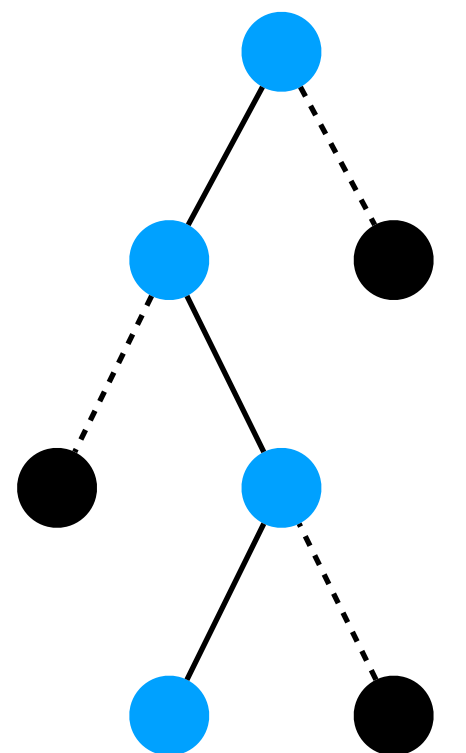
Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
 - Iteratively fixing fractional variables to integer values, solving LP relaxation
 - Goal is to find a feasible integer solution



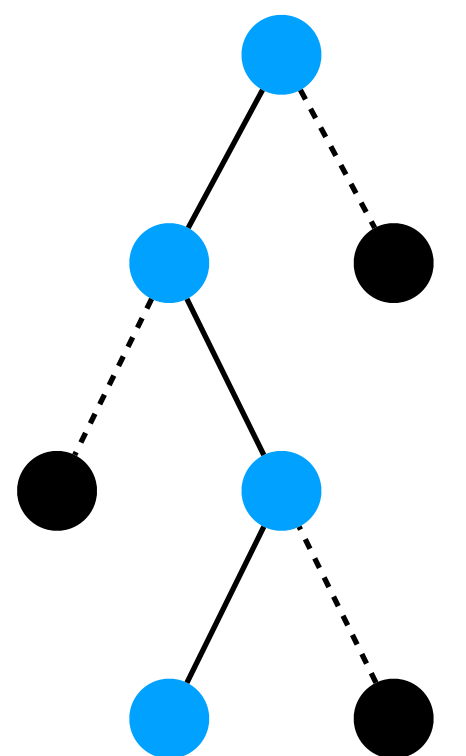
Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
 - Iteratively fixing fractional variables to integer values, solving LP relaxation
 - Goal is to find a feasible integer solution
- Policy choices: **variable order** and **rounding direction**



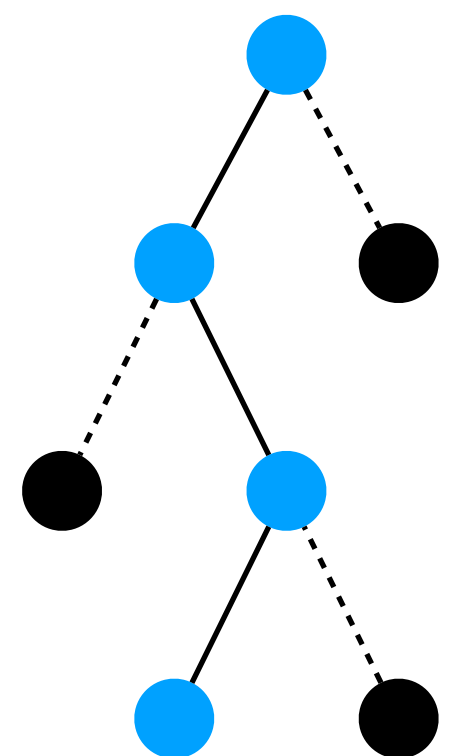
Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
 - Iteratively fixing fractional variables to integer values, solving LP relaxation
 - Goal is to find a feasible integer solution
- Policy choices: **variable order** and **rounding direction**
- **DHEvo** aims to evolve hard instances with improving heuristics



Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
 - Iteratively fixing fractional variables to integer values, solving LP relaxation
 - Goal is to find a feasible integer solution
- Policy choices: **variable order** and **rounding direction**
- **DHEvo** aims to evolve hard instances with improving heuristics
 - Multi-agent LLM generates MILP-heuristic pairs jointly



Outline

1. Background
2. LLMs for speeding up solvers
- 3. Takeaways**

Takeaways

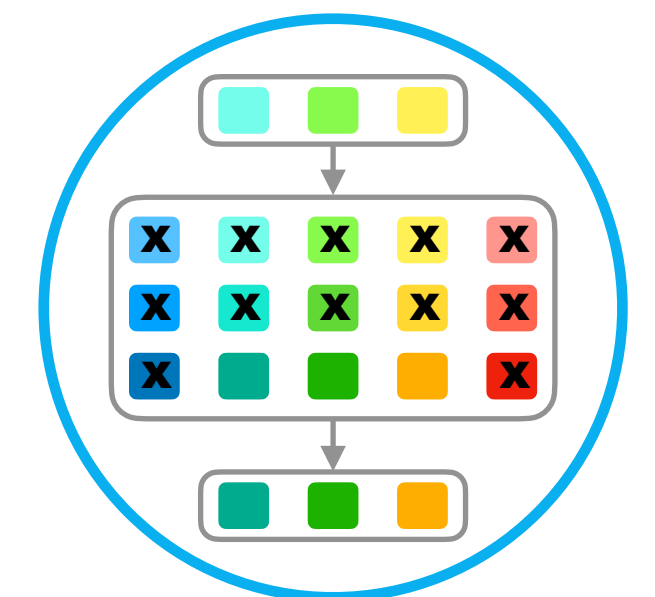
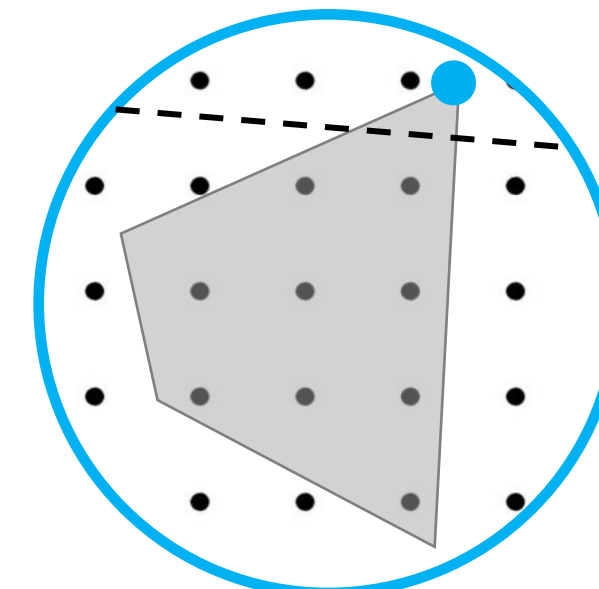
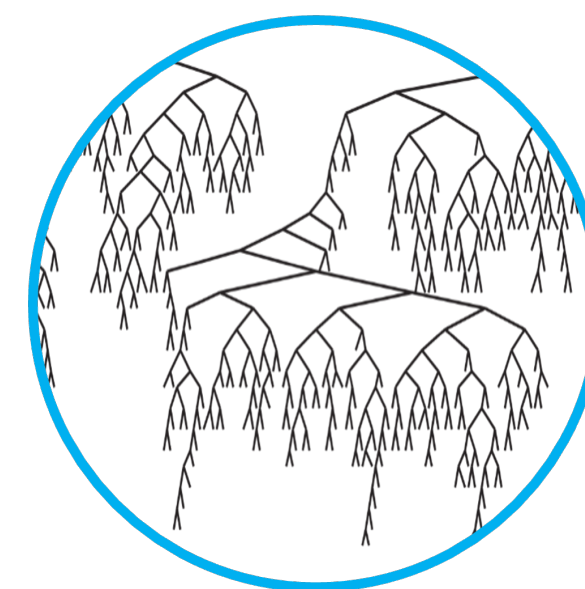
- Solver defaults are rarely optimal: instance families vary widely in structure

Takeaways

- Solver defaults are rarely optimal: instance families vary widely in structure
- **Many** ways to integrate ML into solvers

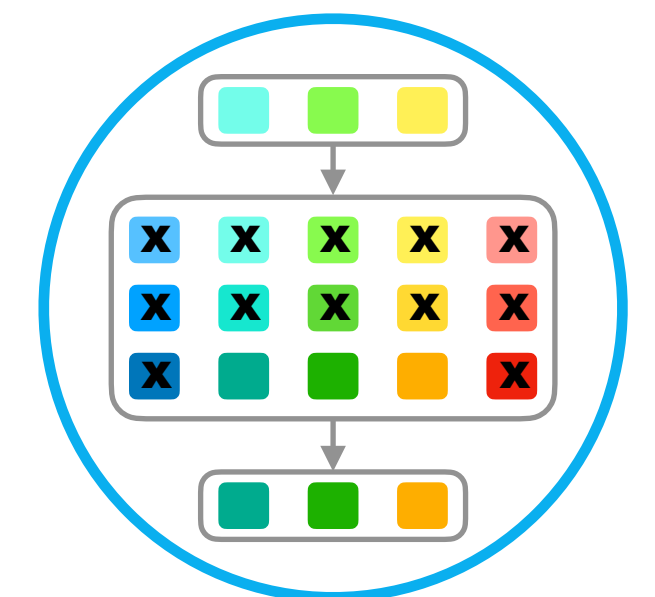
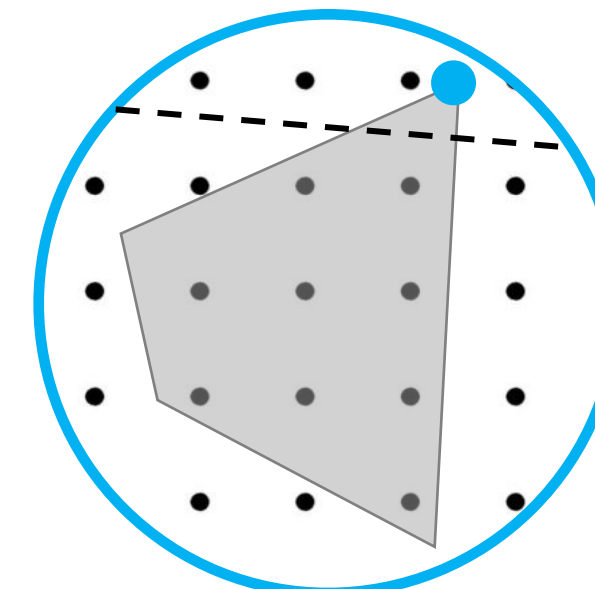
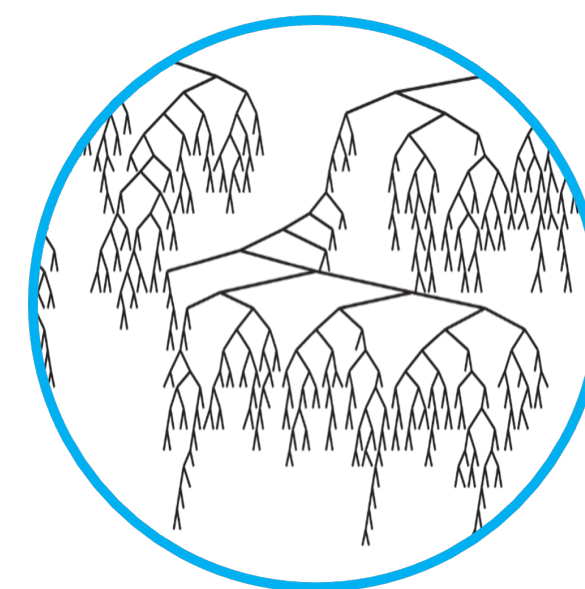
Takeaways

- Solver defaults are rarely optimal: instance families vary widely in structure
- **Many** ways to integrate ML into solvers
- LLMs can be leveraged in many ways to **speed up solvers**



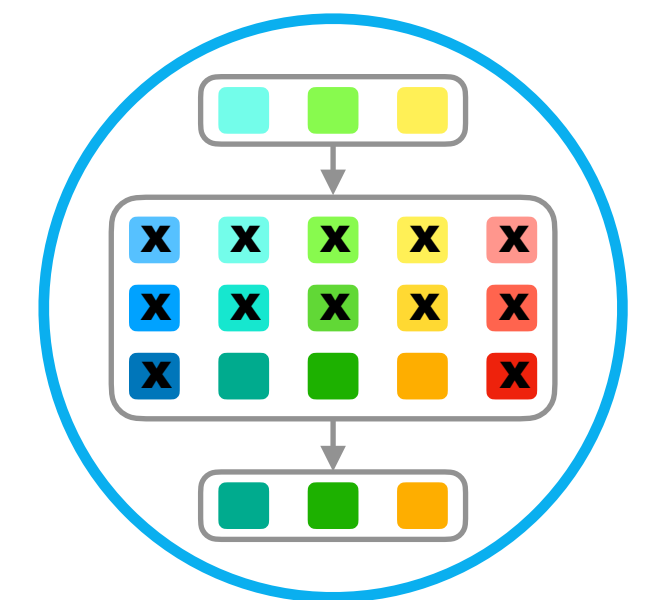
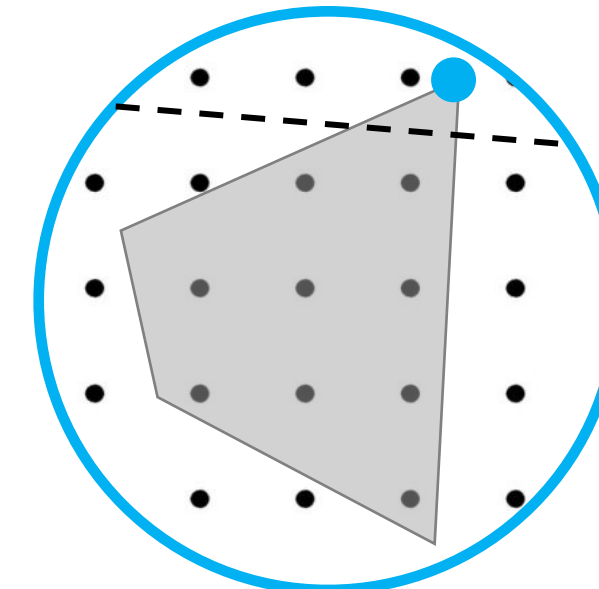
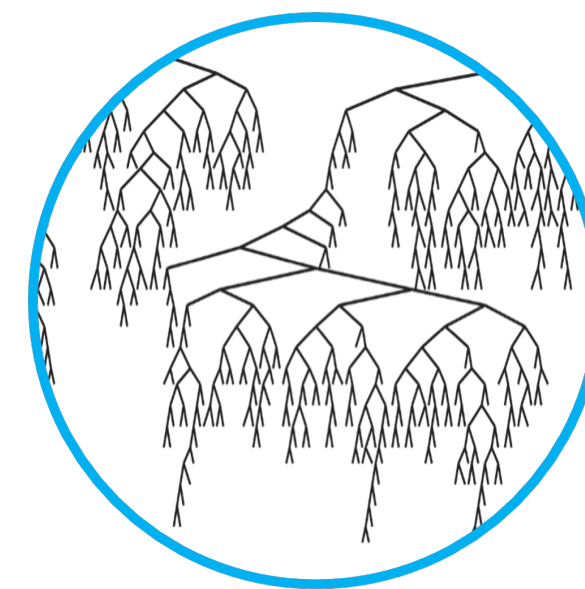
Takeaways

- Solver defaults are rarely optimal: instance families vary widely in structure
- **Many** ways to integrate ML into solvers
- LLMs can be leveraged in many ways to **speed up solvers**
 - **Solver configuration:** LLMs as excellent information retrieval systems



Takeaways

- Solver defaults are rarely optimal: instance families vary widely in structure
- **Many** ways to integrate ML into solvers
- LLMs can be leveraged in many ways to **speed up solvers**
 - **Solver configuration:** LLMs as excellent information retrieval systems
 - **Evolutionary search** for MILP heuristics: LLMs to explore semantic search space
 - Cut selection
 - Large neighborhood search
 - Diving heuristics



Thank you! Questions?

