Northeastern University

MIT LINCOLN LABORATORY

Accelerated Discovery of Set Cover Solutions via Graph Neural Networks

Zohair Shafi
Northeastern University

Benjamin A. Miller
MIT Lincoln Laboratory

Tina Eliassi-Rad
Northeastern University

Rajmonda S. Caceres
MIT Lincoln Laboratory

CPAIOR 2025

Definition

Given a set of elements $\{1,2,\ldots n\}$ called the “Universe” and a collection of subsets, the s**et cover** problem looks at finding the minimum cost subsets whose union “covers” the universe.

The decision version of the set cover problem is NP-complete.

The optimization/search version of the set cover problem is NP-hard.

Set Cover Example

Universe:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Sets:

{1, 2}	**(Cost = 1)**
{3, 4, 5, 6}	**(Cost = 1)**
{7, 8, 9, 10, 11, 12, 13, 14}	**(Cost = 1)**
{1, 3, 5, 7, 9, 11, 13}	**(Cost = 1)**
{2, 4, 6, 8, 10, 12, 14}	**(Cost = 1)**

Set Cover Example

Universe:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Sets:

{1, 2}	**(Cost = 1)**
{3, 4, 5, 6}	**(Cost = 1)**
{7, 8, 9, 10, 11, 12, 13, 14}	**(Cost = 1)**
{1, 3, 5, 7, 9, 11, 13}	**(Cost = 1)**
{2, 4, 6, 8, 10, 12, 14}	**(Cost = 1)**

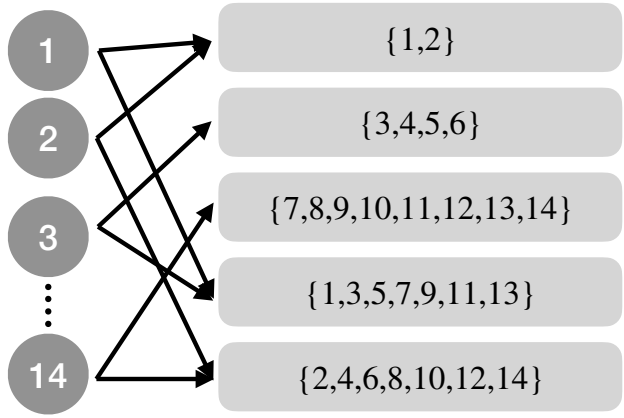
The union of these two subsets is equal to the universe.

Set Cover Example

Universe:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Sets:

{1, 2}	**(Cost = 1)**
{3, 4, 5, 6}	**(Cost = 1)**
{7, 8, 9, 10, 11, 12, 13, 14}	**(Cost = 1)**
{1, 3, 5, 7, 9, 11, 13}	**(Cost = 1)**
{2, 4, 6, 8, 10, 12, 14}	**(Cost = 1)**



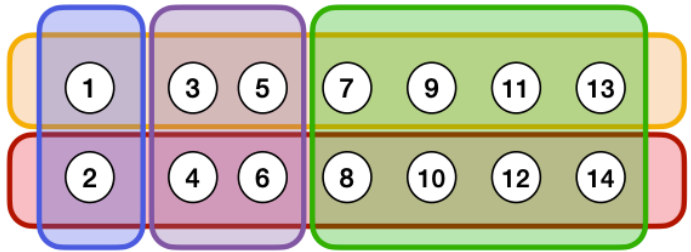
The same problem can be represented as a bipartite graph.

Set Cover Example

Universe:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

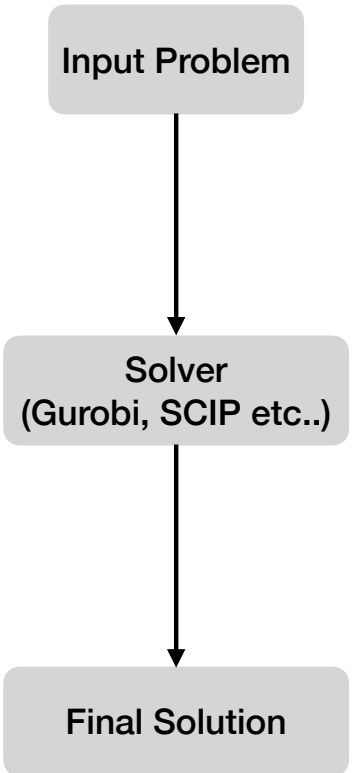
Sets:

{1, 2}	**(Cost = 1)**
{3, 4, 5, 6}	**(Cost = 1)**
{7, 8, 9, 10, 11, 12, 13, 14}	**(Cost = 1)**
{1, 3, 5, 7, 9, 11, 13}	**(Cost = 1)**
{2, 4, 6, 8, 10, 12, 14}	**(Cost = 1)**



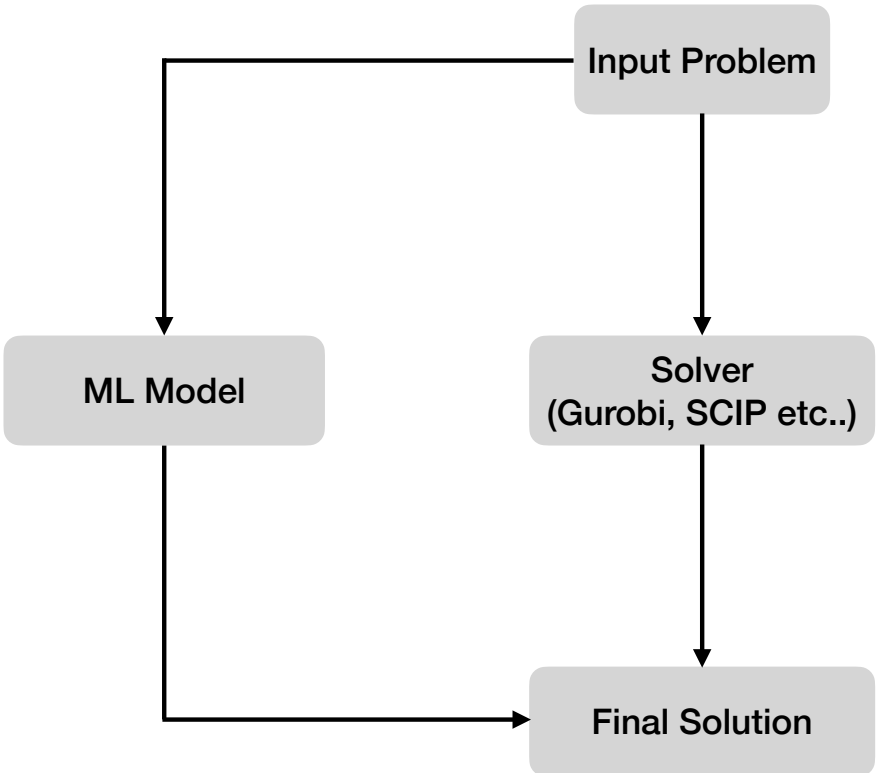
The same problem can also be represented as a hypergraph.

The Big Picture



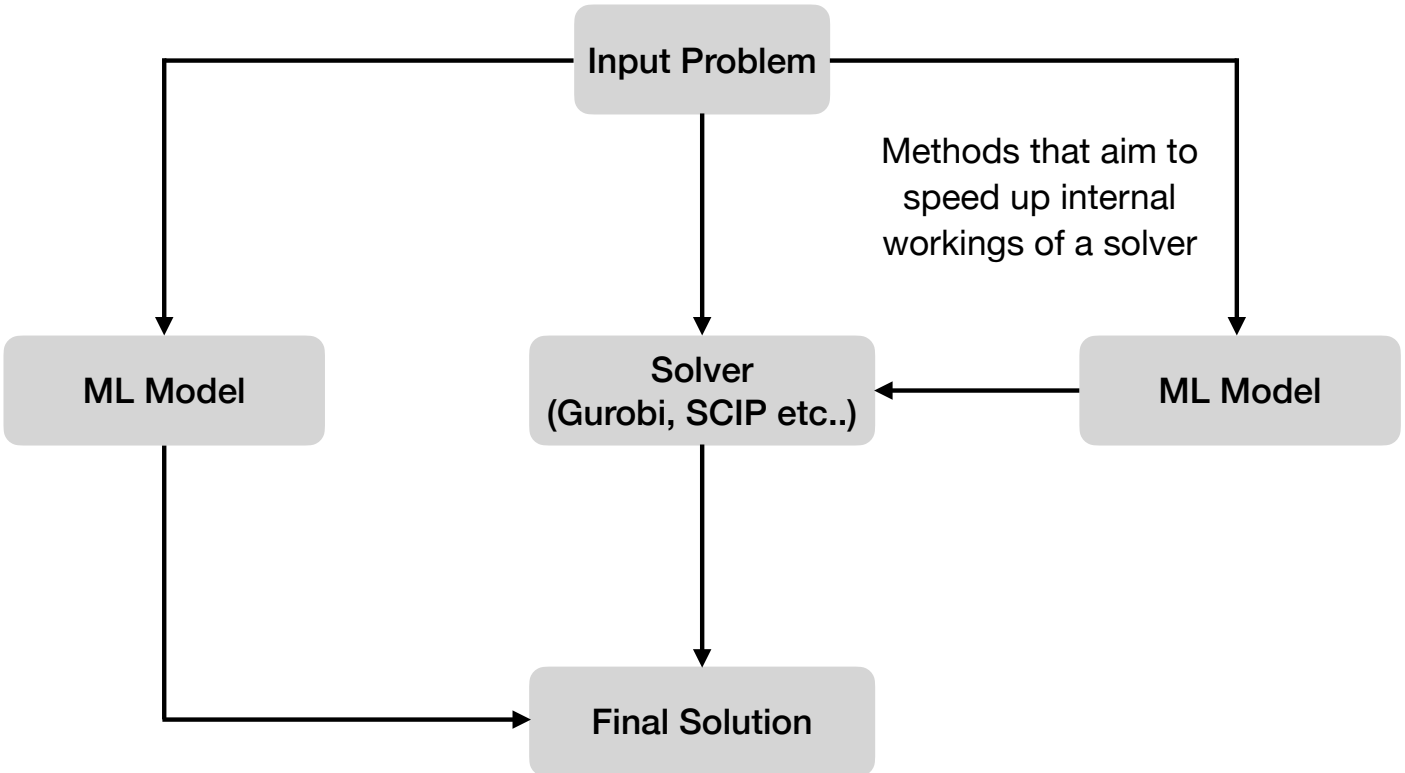
The Big Picture

End-to-end methods that directly approximate the solution



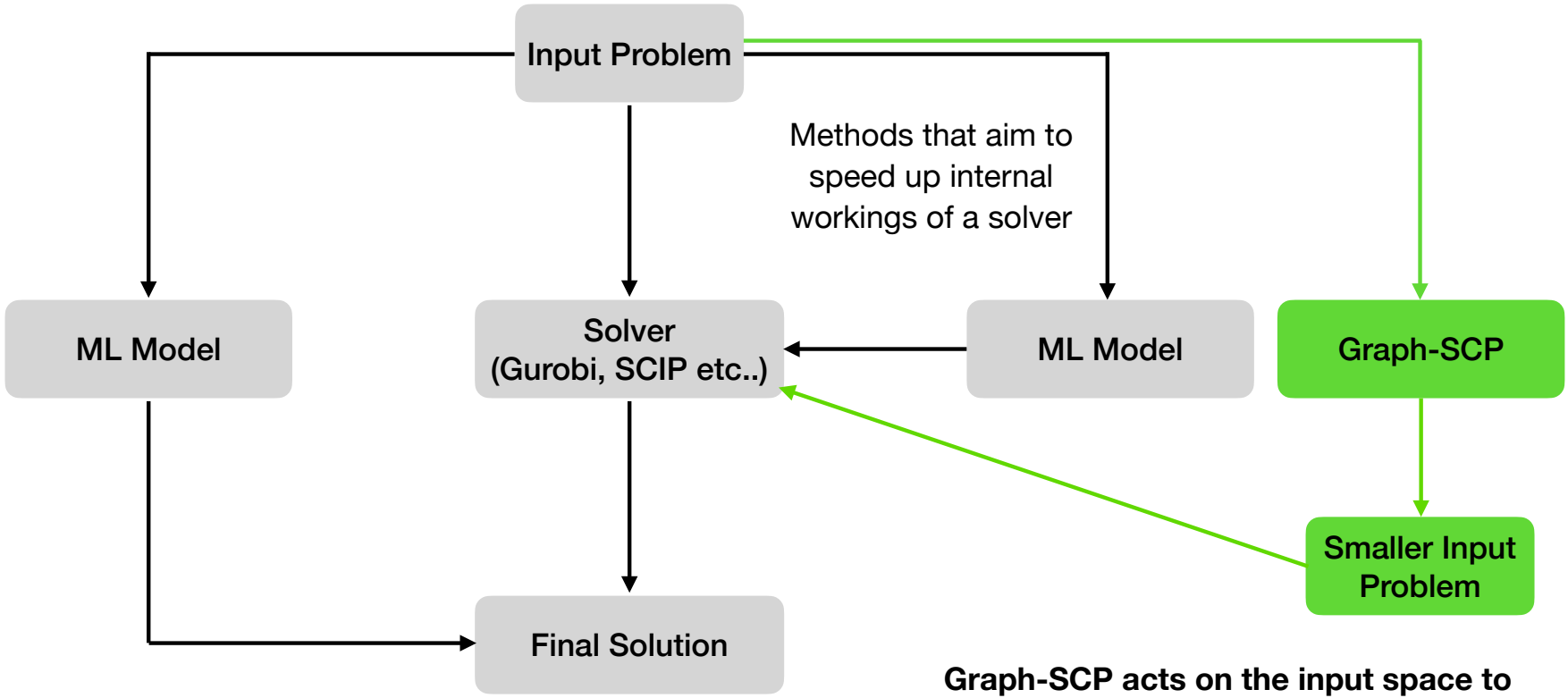
The Big Picture

End-to-end methods that directly approximate the solution



The Big Picture

End-to-end methods that directly approximate the solution

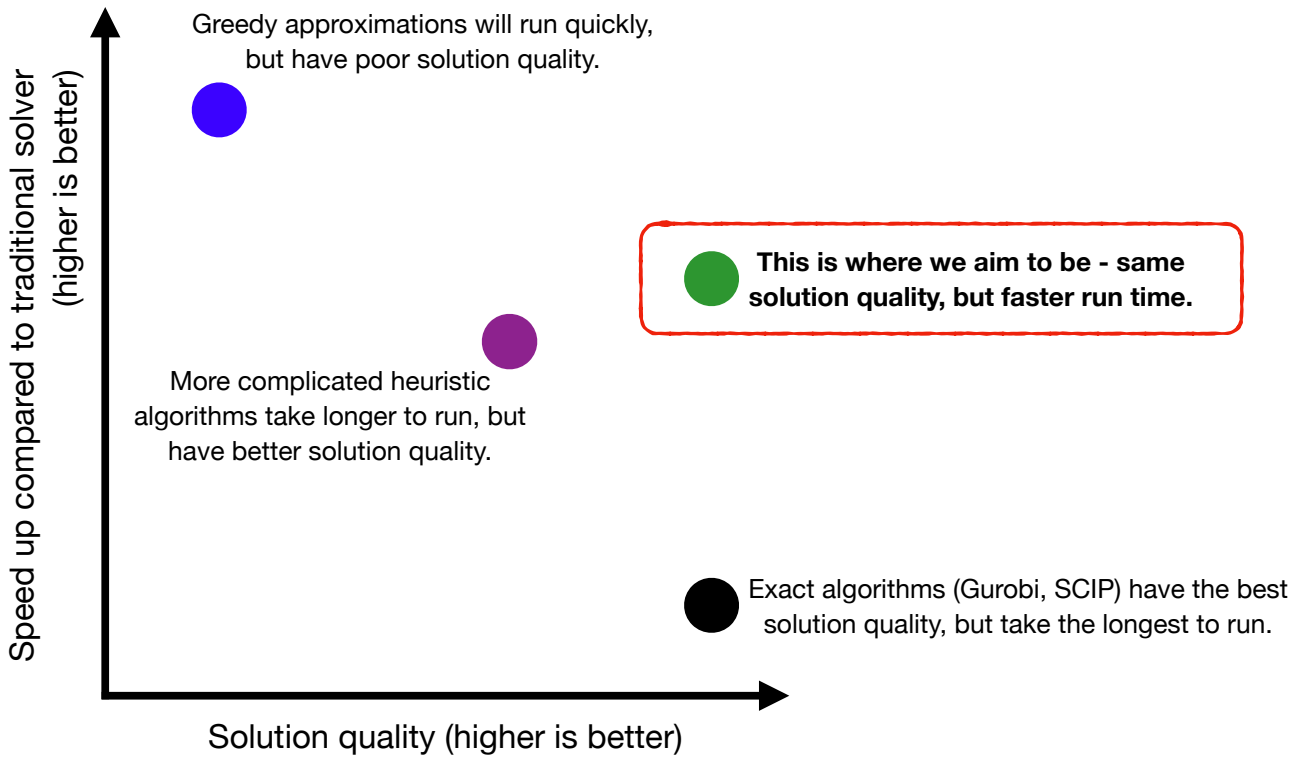


Graph-SCP acts on the input space to produce a smaller problem for the solver.

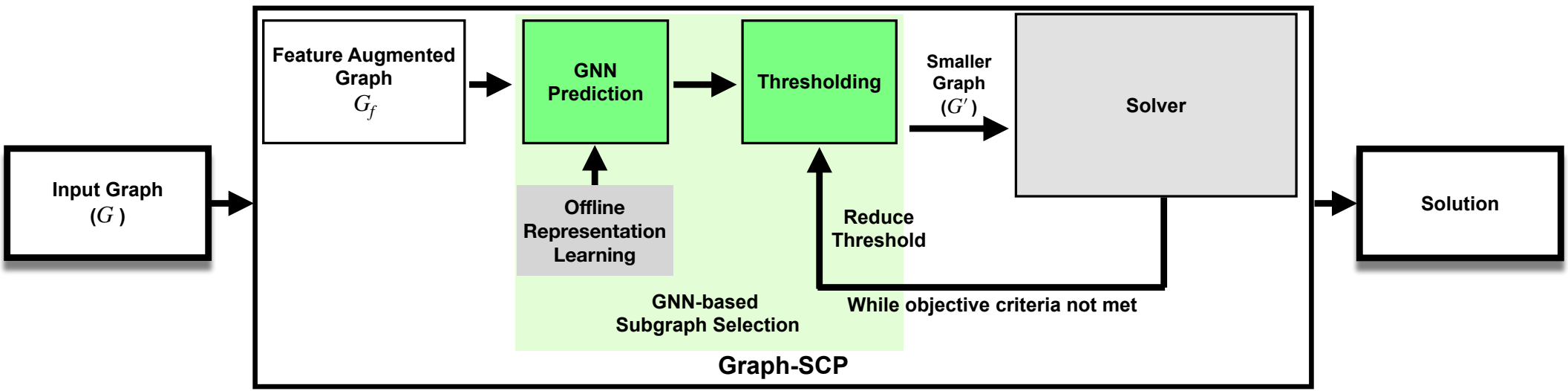
Motivation

- Existing end-to-end methods can run fast, but sacrifice solution quality.
- Can we use graph machine learning to speed up the set cover problem without loss in solution quality?

Motivation

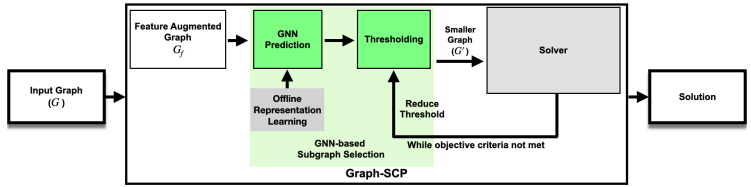

Greedy approximations will run quickly, but have poor solution quality.
This is where we aim to be - same solution quality, but faster run time.
More complicated heuristic algorithms take longer to run, but have better solution quality.
Exact algorithms (Gurobi, SCIP) have the best solution quality, but take the longest to run.
Speed up compared to traditional solver (higher is better)
Solution quality (higher is better)

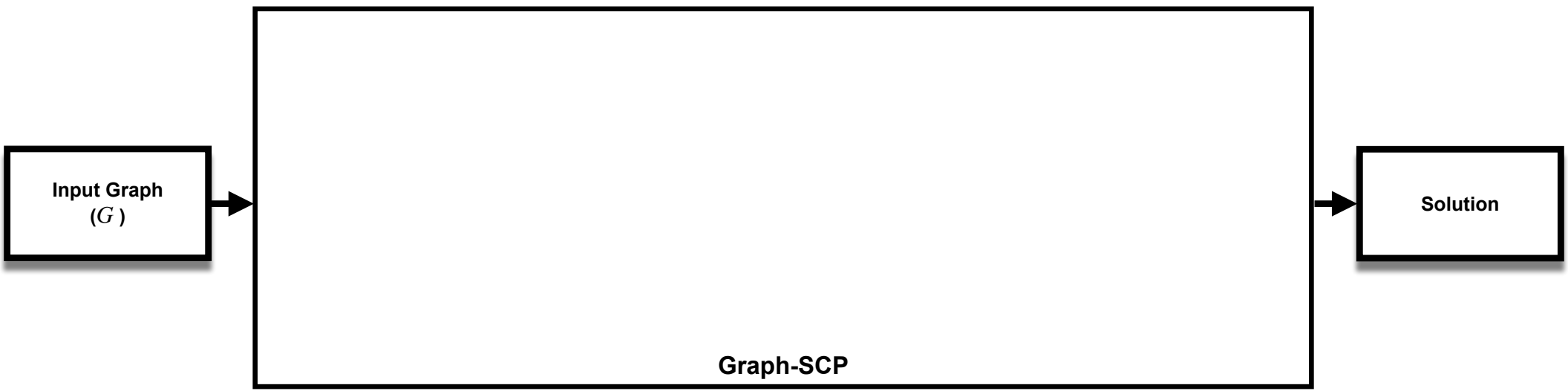
Overview of Graph-SCP



Overview of Graph-SCP

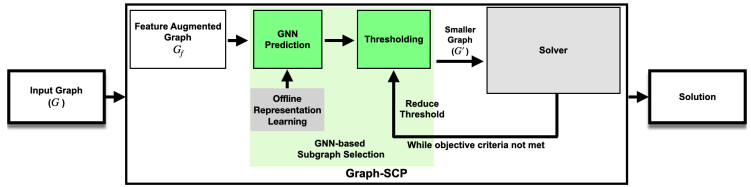
Step-by-step

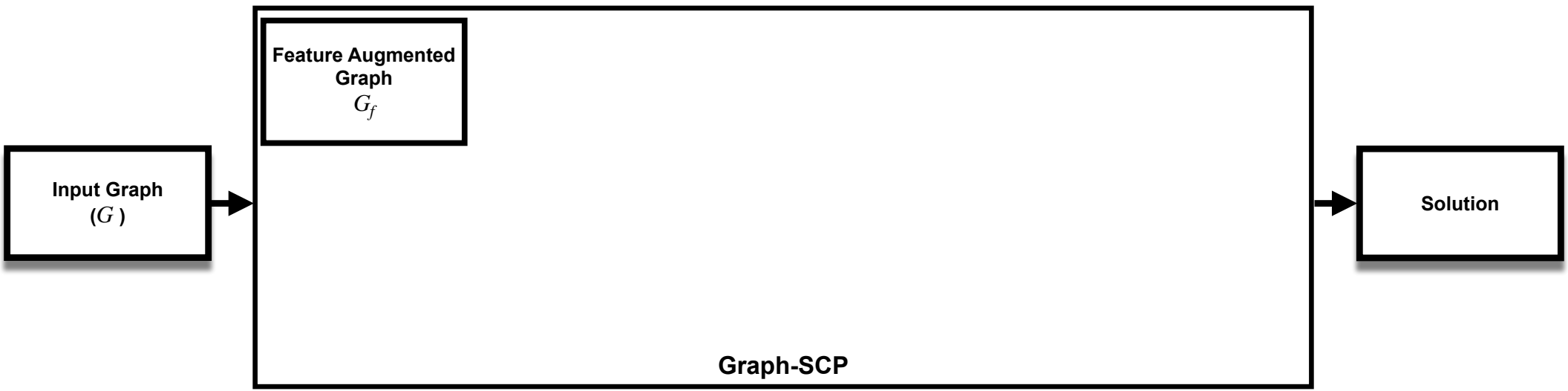




Overview of Graph-SCP

How does it find a subproblem?





Graph-SCP: Node Features

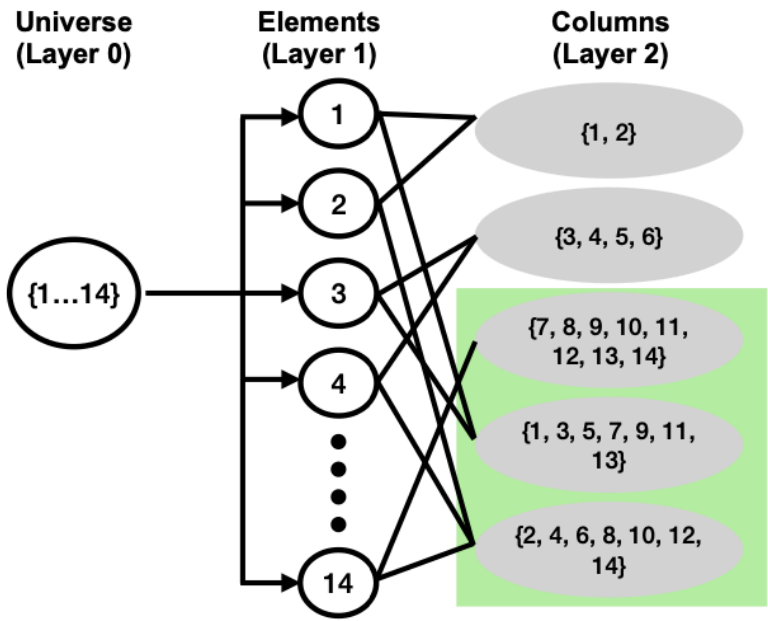
What are the node features?

1. Cost of each subset
2. Cover: number of elements of the universe covered by each subset
3. Random Walks with Restarts (RWR) from the universe node
4. Degree
5. Average Neighbor Degree
6. Hypergraph Features - eigen values of the Laplacian of the hypergraph random walk matrix

Graph-SCP: Node Features

What are the node features?

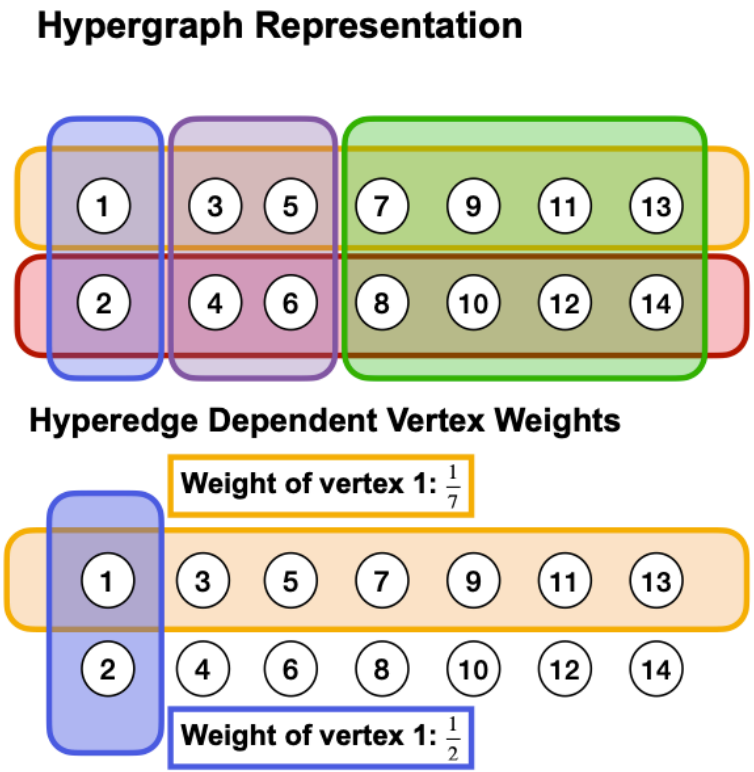
1. Cost of each subset
2. Cover: number of elements of the universe covered by each subset
3. Random Walks with Restarts (RWR) from the universe node
4. Degree
5. Average Neighbor Degree
6. Hypergraph Features - eigen values of the Laplacian of the hypergraph random walk matrix



Graph-SCP: Node Features

What are the node features?

1. Cost of each subset
2. Cover: number of elements of the universe covered by each subset
3. Random Walks with Restarts (RWR) from the universe node
4. Degree
5. Average Neighbor Degree
6. Hypergraph Features - eigen values of the Laplacian of the hypergraph random walk matrix



Graph-SCP: Training Data

What does the data look like?

Instance	m	n	d	Cost
Type 1	100-400	100-1000	0.22-0.29	Uniform [100-200]
Type 2	100-300	100-500	0.16-0.28	Equal
Type 3	200-350	300-350	0.13-0.18	Poisson ($\lambda = 20$)
Type 4	200-250	1000-3000	0.04-0.05	Poisson ($\lambda = 20$)
Type 5 (OR Library)	100-500	1000-10000	0.02-0.2	Uniform [1-100]

Table 1: Characteristics of SCP instances used during training and testing of Graph-SCP. Here, m is the number of rows, n the number of columns, and d is the density of the instance. Further details of the sets within the OR Library and their results are shown in Table 2.

Graph-SCP: Training Data

What does the data look like?

Progressively takes longer to solve

Instance	m	n	d	Cost
Type 1	100-400	100-1000	0.22-0.29	Uniform [100-200]
Type 2	100-300	100-500	0.16-0.28	Equal
Type 3	200-350	300-350	0.13-0.18	Poisson ($\lambda = 20$)
Type 4	200-250	1000-3000	0.04-0.05	Poisson ($\lambda = 20$)
Type 5 (OR Library)	100-500	1000-10000	0.02-0.2	Uniform [1-100]

Table 1: Characteristics of SCP instances used during training and testing of Graph-SCP. Here, m is the number of rows, n the number of columns, and d is the density of the instance. Further details of the sets within the OR Library and their results are shown in Table 2.

Graph-SCP: Training Data

What does the data look like?

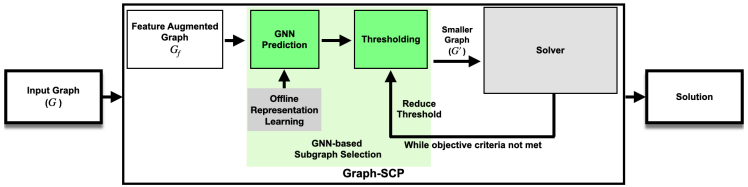
Instance	m	n	d	Cost
Type 1	100-400	100-1000	0.22-0.29	Uniform [100-200]
Type 2	100-300	100-500	0.16-0.28	Equal
Type 3	200-350	300-350	0.13-0.18	Poisson ($\lambda = 20$)
Type 4	200-250	1000-3000	0.04-0.05	Poisson ($\lambda = 20$)
Type 5 (OR Library)	100-500	1000-10000	0.02-0.2	Uniform [1-100]

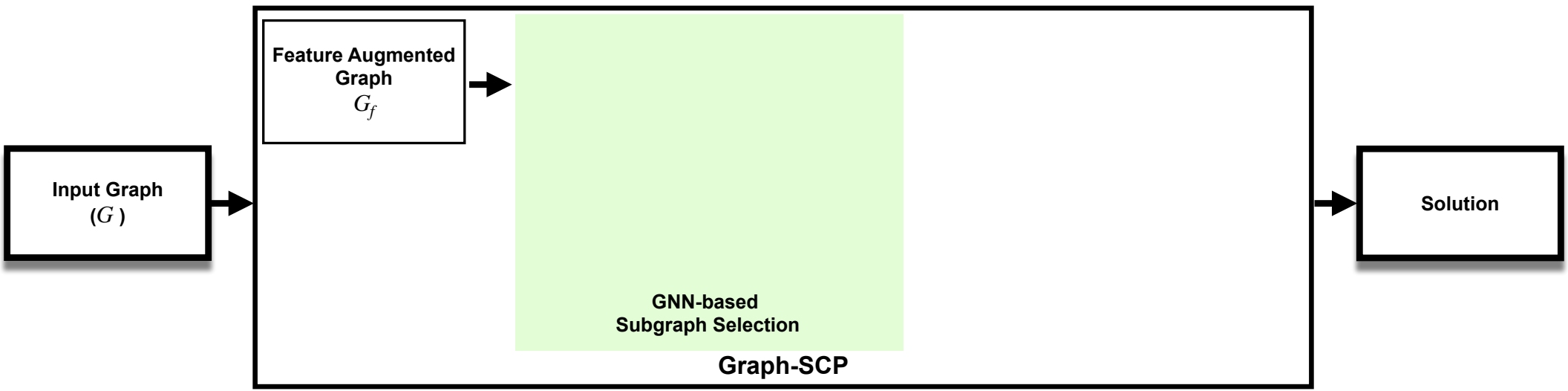
Used as test set only

Table 1: Characteristics of SCP instances used during training and testing of Graph-SCP. Here, m is the number of rows, n the number of columns, and d is the density of the instance. Further details of the sets within the OR Library and their results are shown in Table 2.

Overview of Graph-SCP

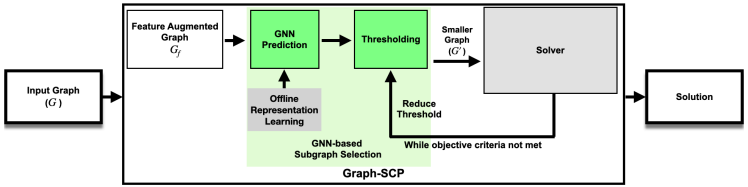
How does it find a subproblem?

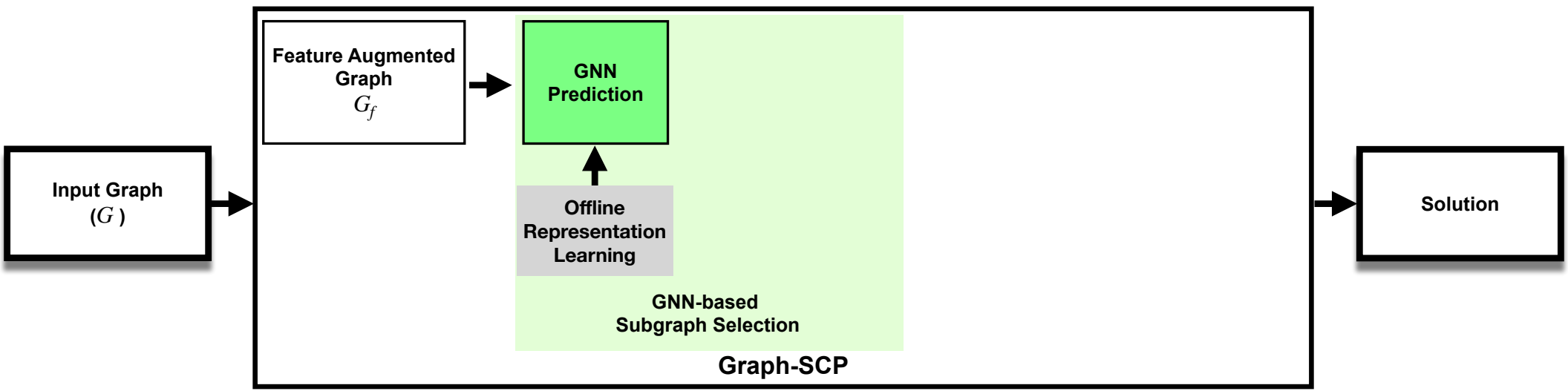




Overview of Graph-SCP

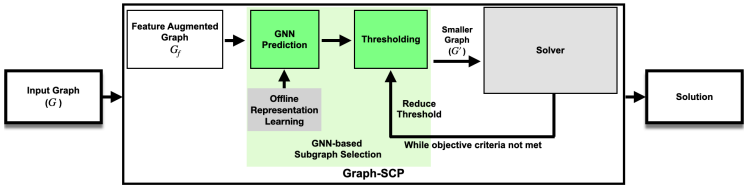
How does it find a subproblem?

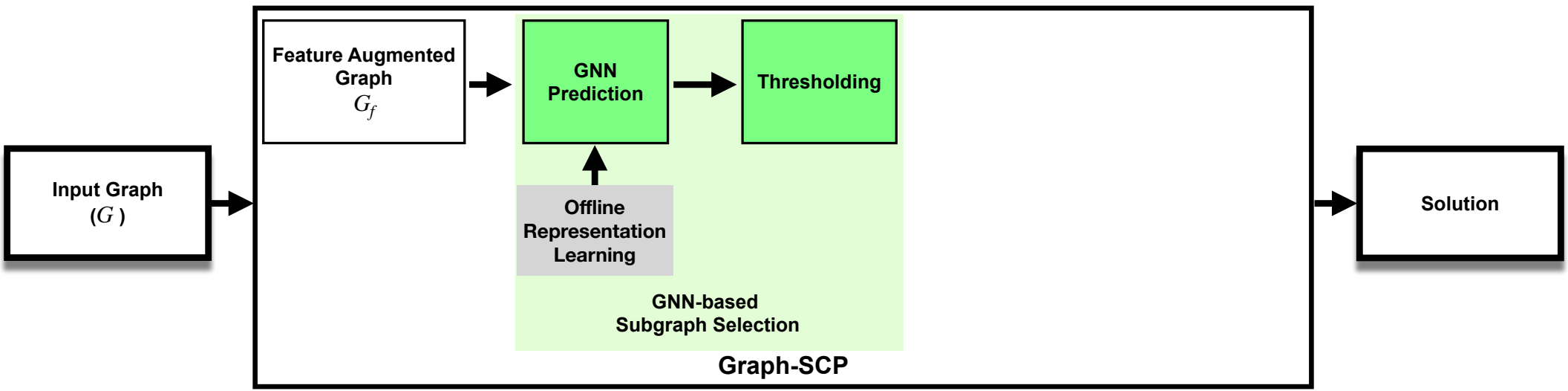




Overview of Graph-SCP

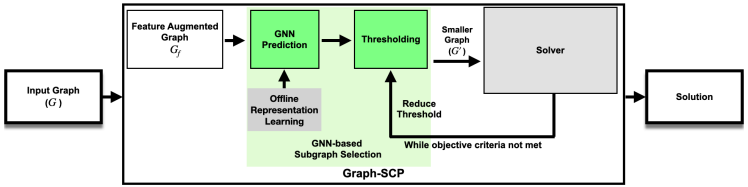
How does it find a subproblem?

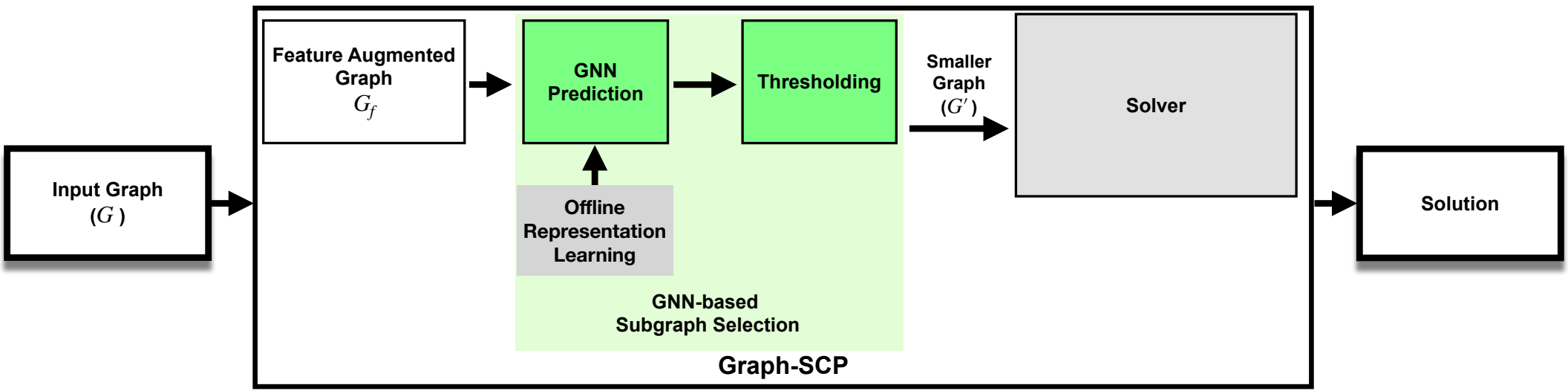




Overview of Graph-SCP

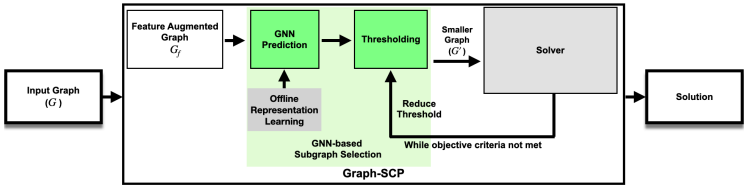
How does it find a subproblem?





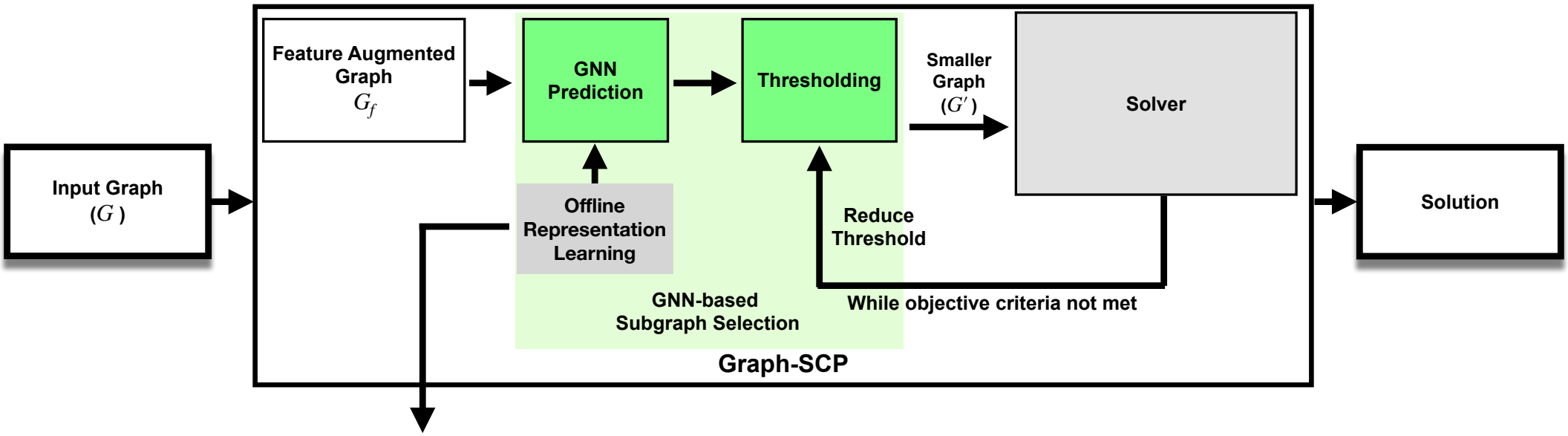
Overview of Graph-SCP

How does it find a subproblem?



Overview of Graph-SCP

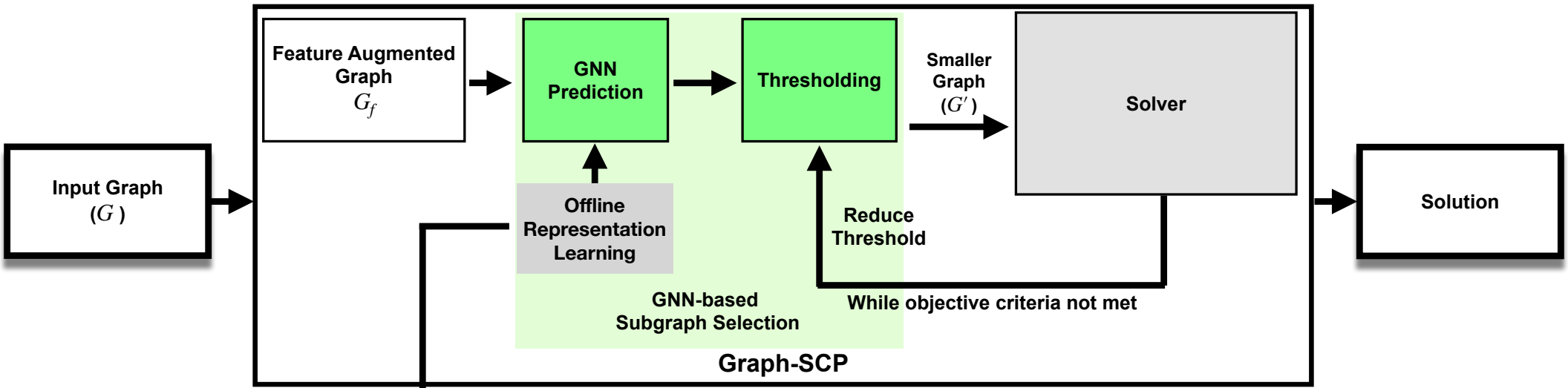
Objective Function



$$L(y, \hat{y}) = \alpha(y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})) \; + \beta[\sum (\hat{y} \cdot c)^2 - \gamma \sum (A\hat{y} - 1) - \omega \sum (1 - A\hat{y})]$$

Overview of Graph-SCP

Objective Function

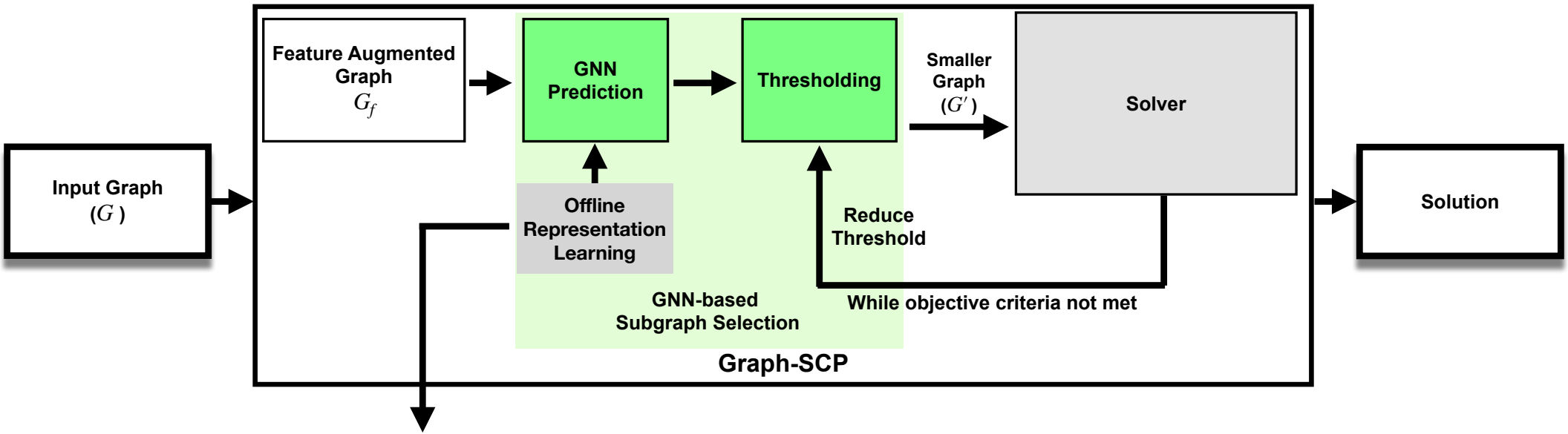


$$L(y,\hat{y}) = \alpha(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) + \beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})]$$

Binary Cross Entropy

Overview of Graph-SCP

Objective Function

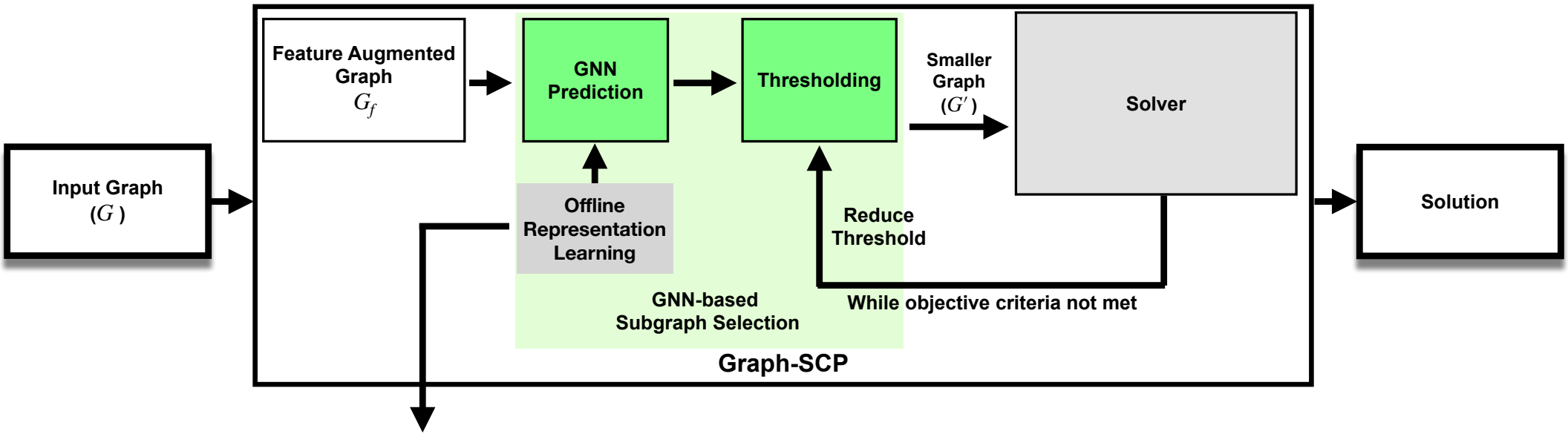


$$L(y, \hat{y}) = \alpha(y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})) + \beta[\sum (\hat{y} \cdot c)^2 - \gamma \sum (A\hat{y} - 1) - \omega \sum (1 - A\hat{y})]$$

Minimize Linear Program
Relaxation of Set Cover

Overview of Graph-SCP

Objective Function

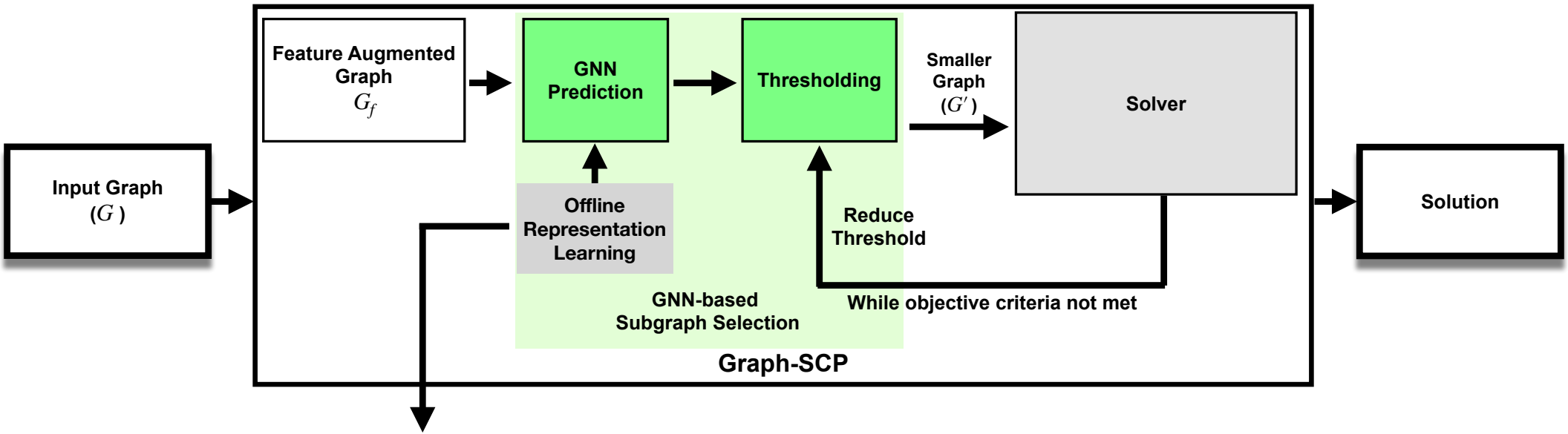


$$L(y,\hat{y}) = \alpha(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) \; + \beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})]$$

Penalize Cost &
Number of Sets Picked

Overview of Graph-SCP

Objective Function

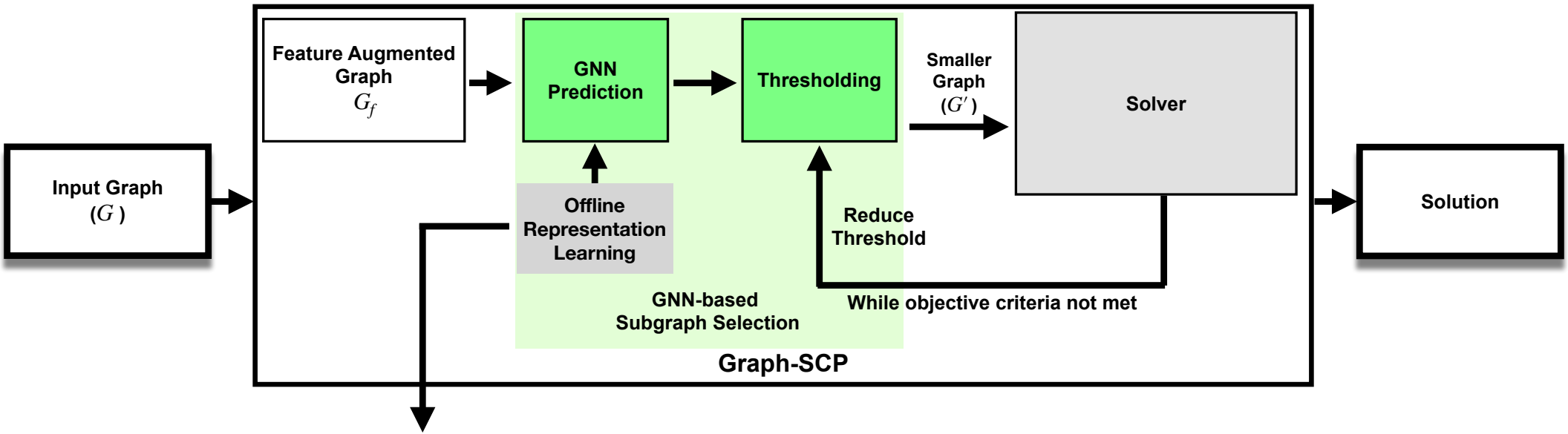


$$L(y,\hat{y}) = \alpha(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) \; + \beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})]$$

Penalize Unsatisfied Constraints

Overview of Graph-SCP

Objective Function

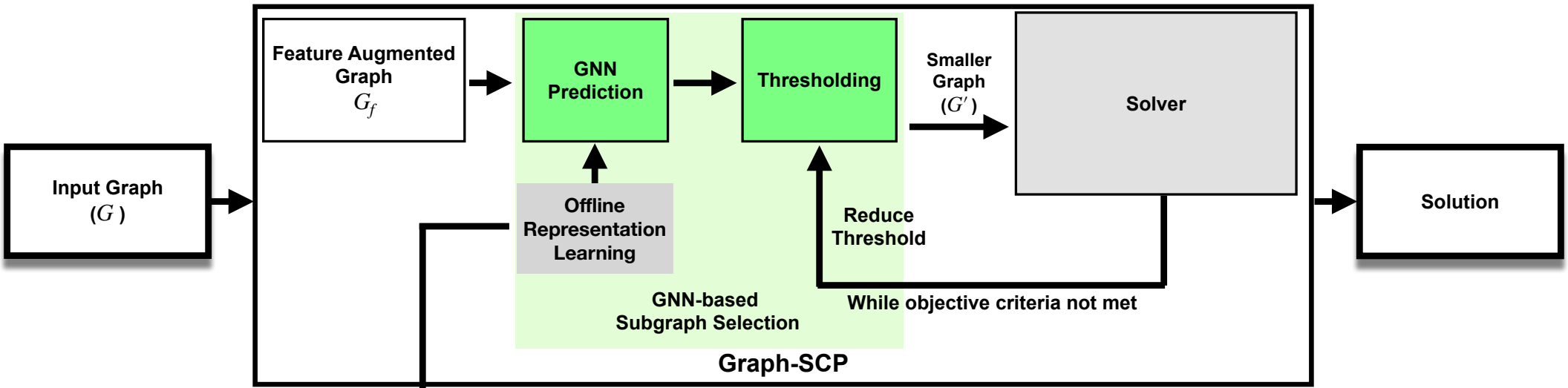


$$L(y,\hat{y}) = \alpha(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) \; + \beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})]$$

Encourage Sparse Selection of Sets

Overview of Graph-SCP

Objective Function



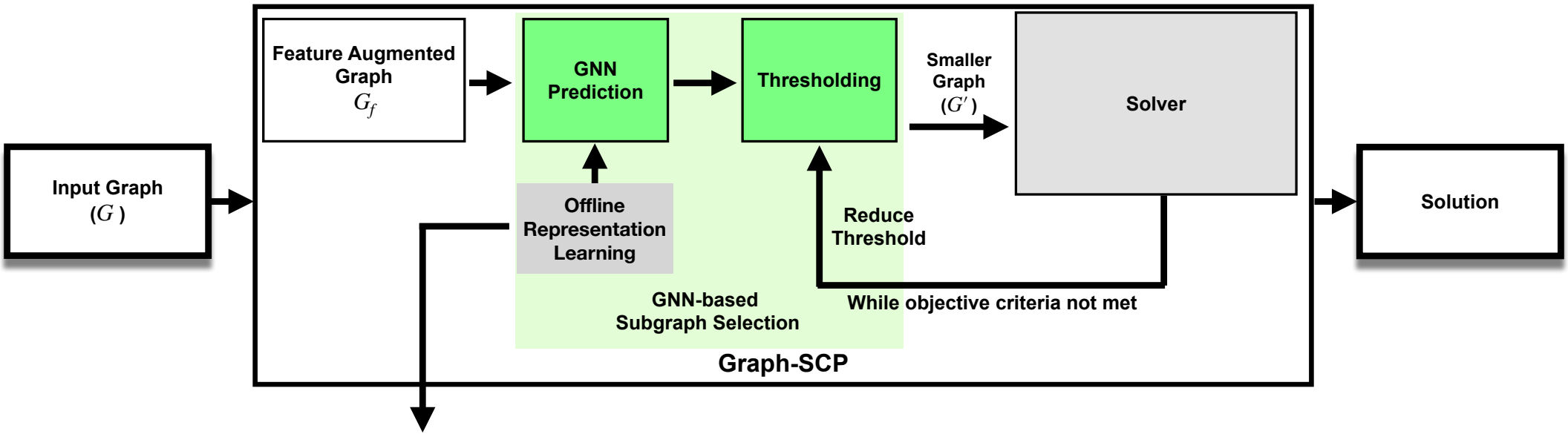
$$L(y, \hat{y}) = \alpha(y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})) + \beta[\sum (\hat{y} \cdot c)^2 - \gamma \sum (A\hat{y} - 1) - \omega \sum (1 - A\hat{y})]$$

Hyperparameter for Supervised Portion of Loss

Hyperparameter for Unsupervised Portion of Loss

Overview of Graph-SCP

Objective Function

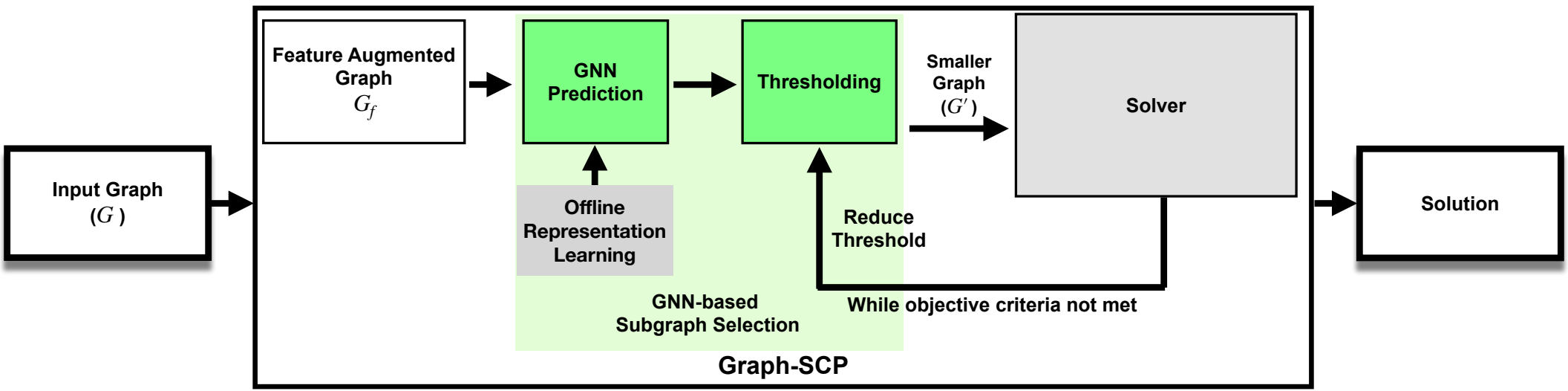


$$L(y,\hat{y}) = \alpha(y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})) \; + \beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})]$$

Hyperparameters Inside
Unsupervised Portion of Loss

Overview of Graph-SCP

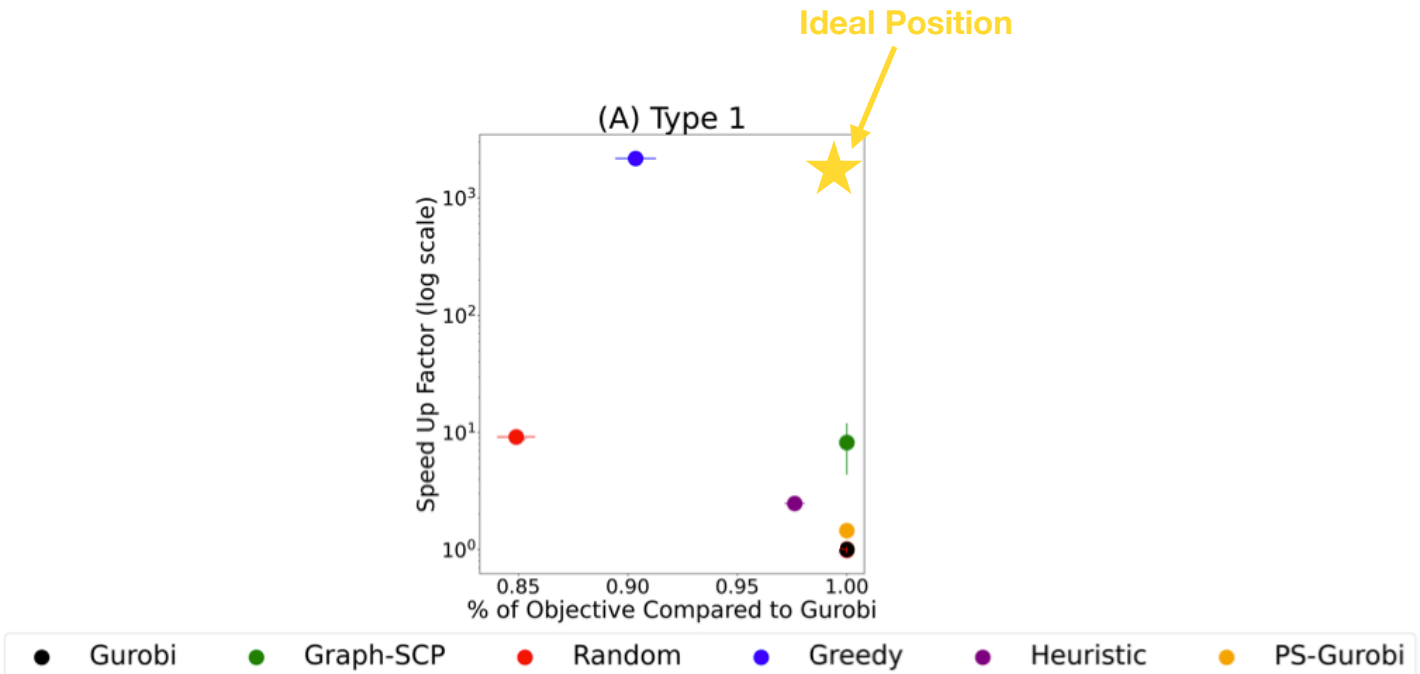
Final Picture



Our experiments show…

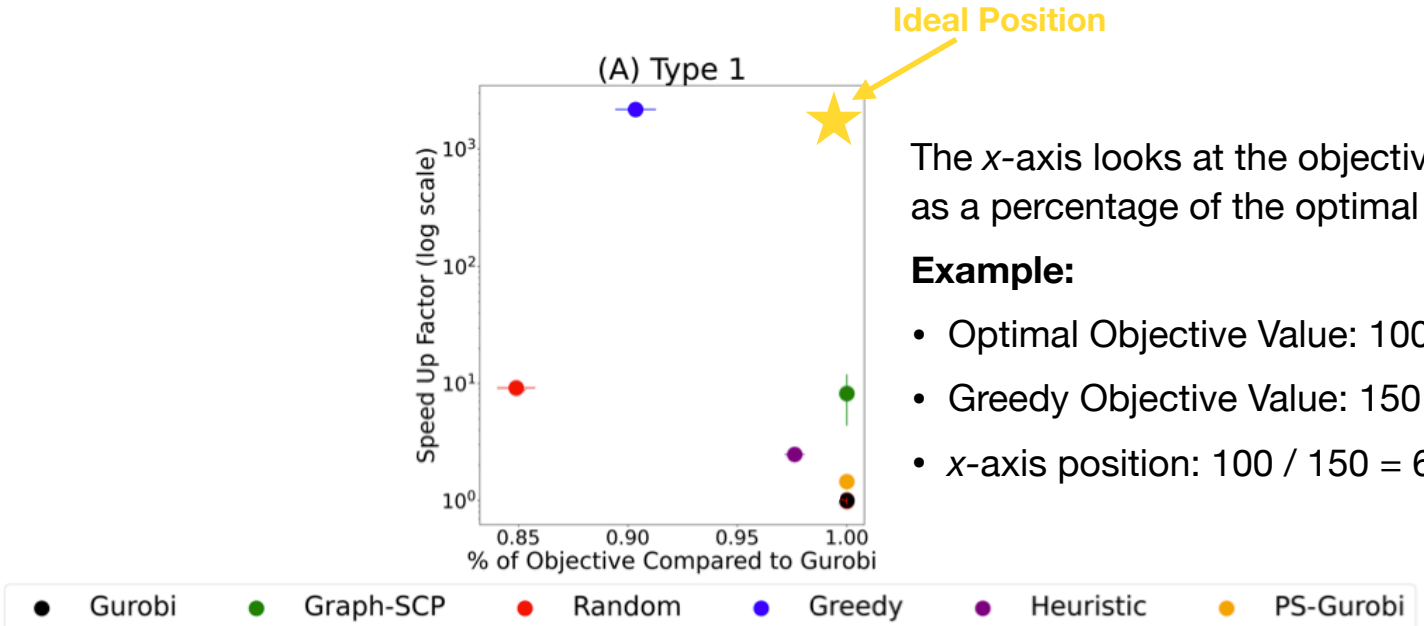
1. Graph-SCP achieves optimal objective values with faster run times.
2. Graph-SCP works well on unseen distribution of instances.
   - It achieves an average speedup of 3.2x on instances from the OR Library.
3. Graph-SCP achieves lower primal gaps.

Graph-SCP achieves optimal objective values with faster run times



Graph-SCP achieves optimal objective values with faster run times

We care about the *x*-axis (% of objective compared to Gurobi) more than the *y*-axis (speedup factor).



The *x*-axis looks at the objective achieved as a percentage of the optimal (i.e., Gurobi).

Example:

- Optimal Objective Value: 100
- Greedy Objective Value: 150
- *x*-axis position: 100 / 150 = 66.7%

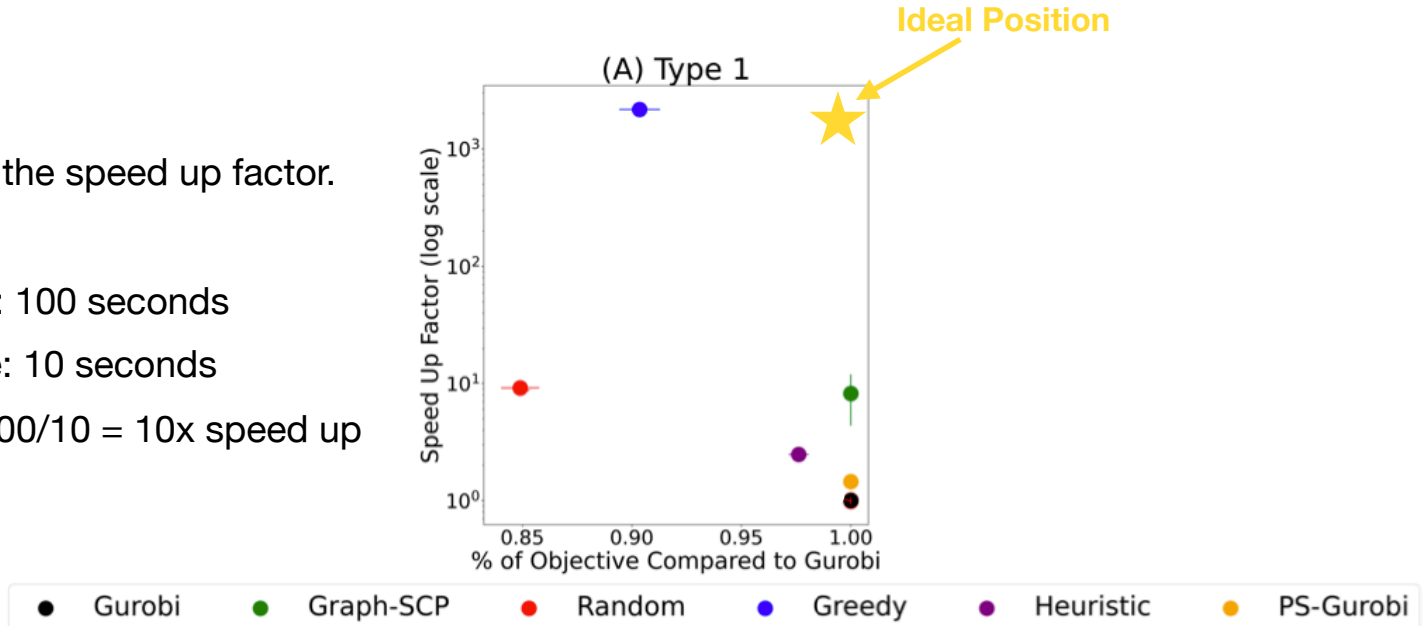
Graph-SCP achieves optimal objective values with faster run times

We care about the *x*-axis (% of objective compared to Gurobi) more than the *y*-axis (speed-up factor).

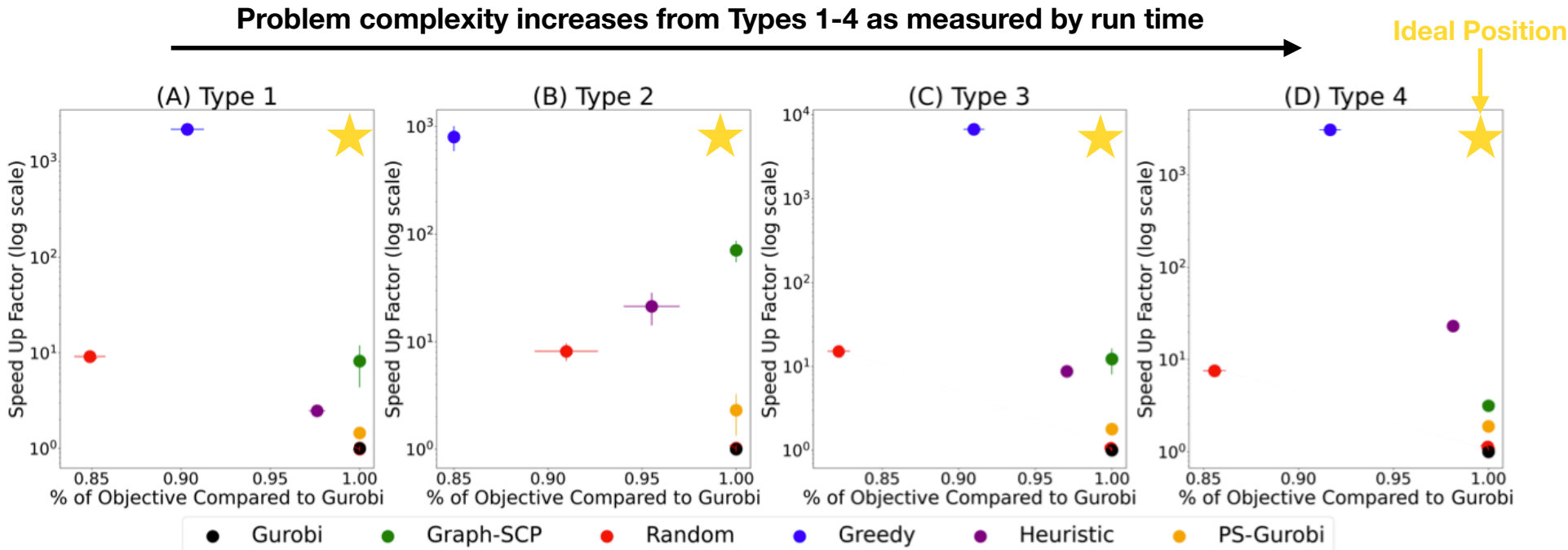
The *y*-axis looks at the speed up factor.

Example:

- Gurobi Run Time: 100 seconds
- Greedy Run Time: 10 seconds
- *y*-axis position: 100/10 = 10x speed up



Graph-SCP achieves optimal objective values with faster run times



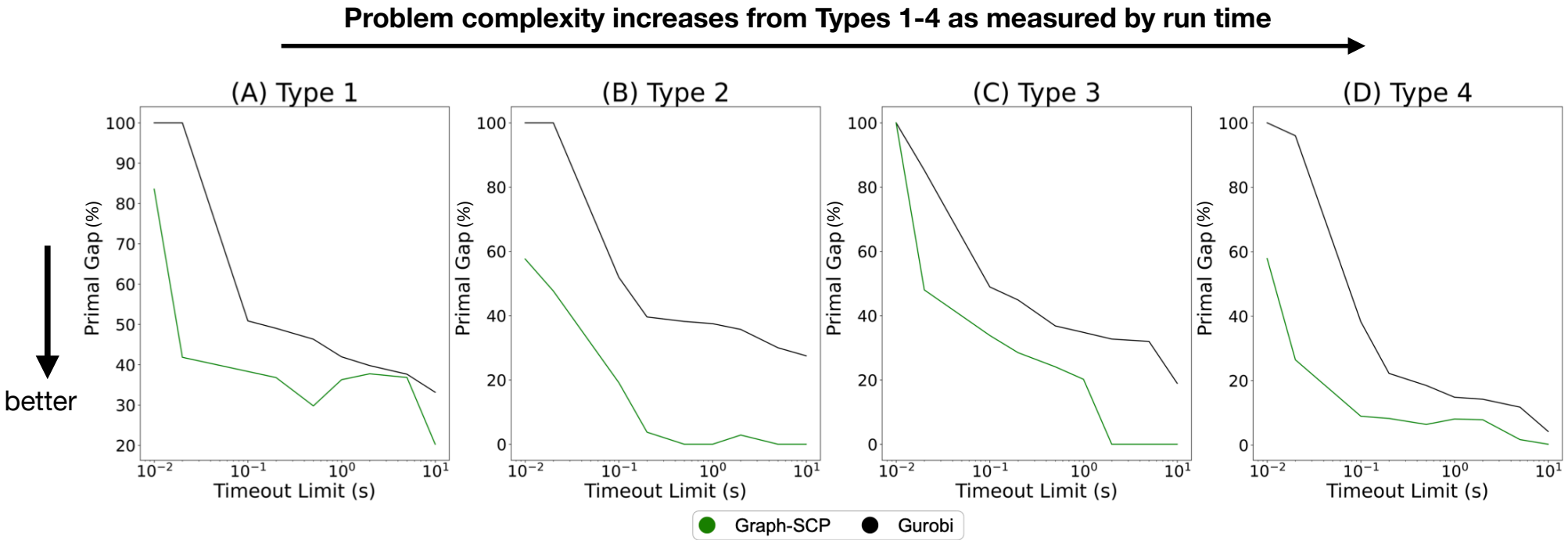
Graph-SCP achieves an average speedup of 3.2x on instances from the OR Library

Progressively gets harder to solve w.r.t. run time

Set	# Instances	m	n	Cost	d	Speedup Factor	Mean Runtime Gurobi (s)	Size Reduction
6	5	200	1000	1–100	0.05	1.26	0.10	75%
A	5	300	3000	1–100	0.02	1.28	0.22	84%
B	5	300	3000	1–100	0.05	2.34	0.97	89%
C	5	400	4000	1–100	0.02	2.19	0.69	83%
D	5	400	4000	1–100	0.05	3.48	2.92	86%
NRE	5	500	5000	1–100	0.1	5.96	36.90	94%
NRF	5	500	5000	1–100	0.2	8.15	32.6	96%
NRG	5	1000	10000	1–1000	0.02	1.17	2003.27	71%

Table 2: Results for the sets of instances defined in the OR Library (Beasley 1990). Each set (A, B, NRE etc.) has its own definitive characteristics. Here, m is the number of rows, n the number of columns and d is the density of the instance. Graph-SCP has an average speedup of 3.2x across all sets while achieving the optimal objective and an average of approximately 84% reduction in problem size.

Graph-SCP achieves lower primal gaps



Graph-SCP: Takeaway points

1. Graph-SCP finds a sub-problem for a given set cover problem
2. Graph-SCP uses features computed from the graph and hypergraph representations of the set cover problem as input features.
3. Graph-SCP achieves **an average run time speedup of ~10x while maintaining solution quality compared to Gurobi**.

Thank you!