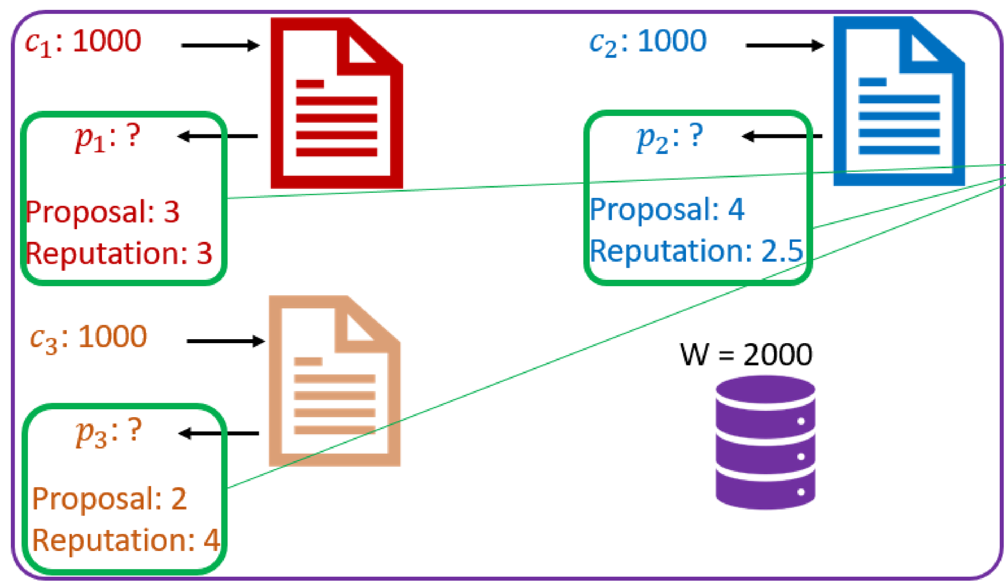
Branch & Learn for Recursively and Iteratively Solvable Problems in Predict+Optimize

Xinyi Hu [1], Jasper C.H. Lee [2], Jimmy H.M. Lee [1], Allen Z. Zhong [1]
[1] The Chinese University of Hong Kong
[2] University of Wisconsin–Madison

NEURAL INFORMATION PROCESSING SYSTEMS

Motivation

In practice, constraint optimization problems (COPs) usually contain parameters, some of which may be unknown, but with related features and historical data.



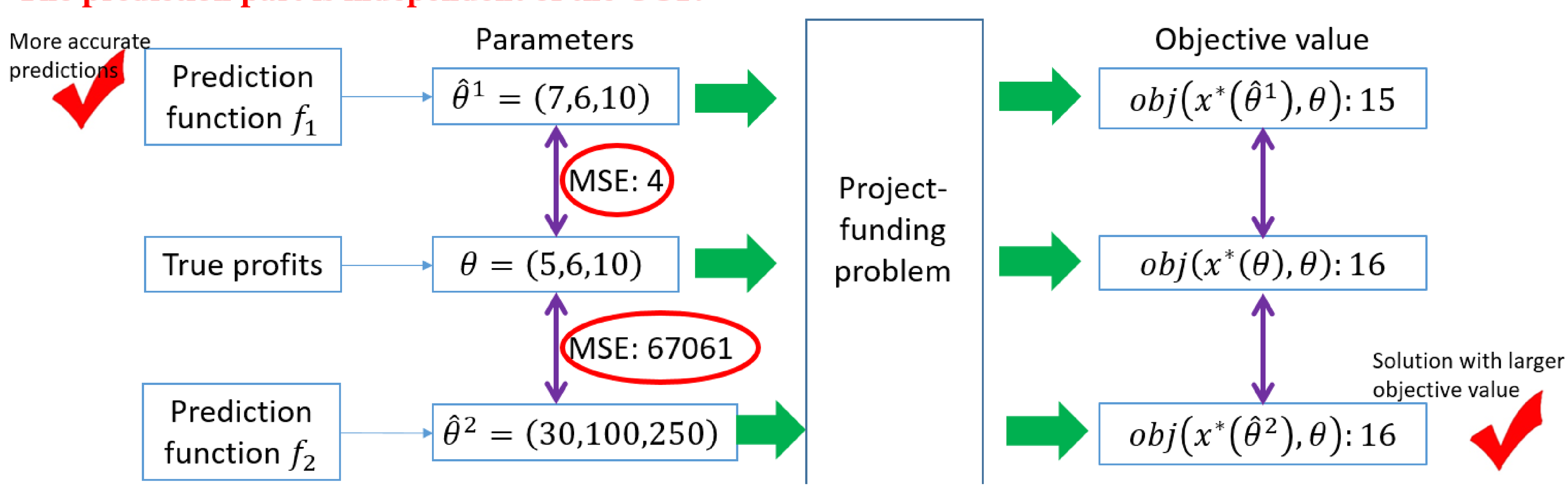
Project funding problem

Unknown profits
Known related features
- proposal scores
- the reputation of each applicant

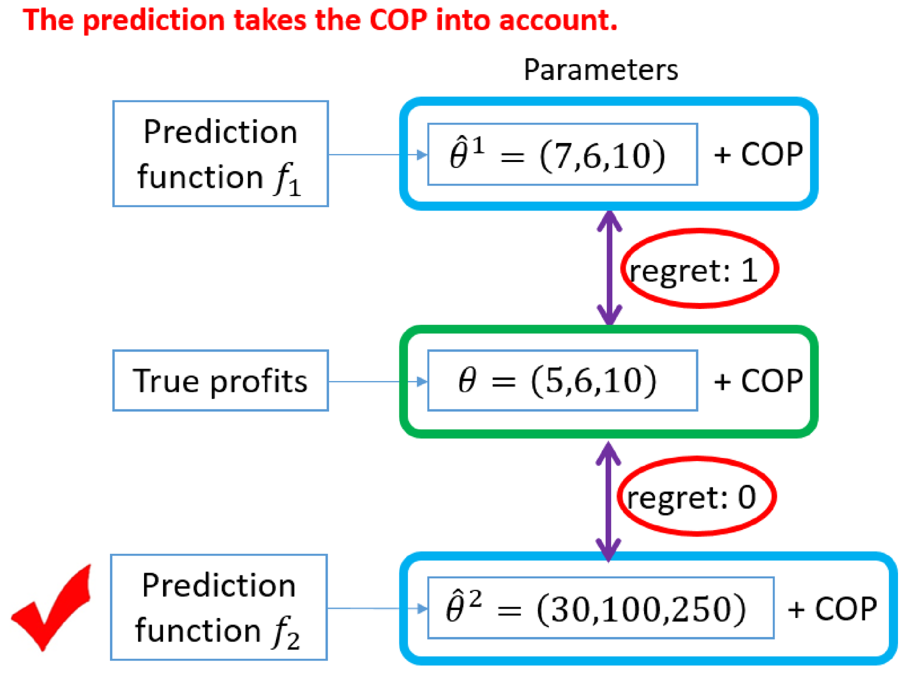
Optimal solution: $\{x_1=?, x_2=?, x_3=?\}$

Classical approaches: first predict, then optimize.
The prediction part is independent of the COP.

When employed in the optimization problem, a "good" forecast may give a poor result.

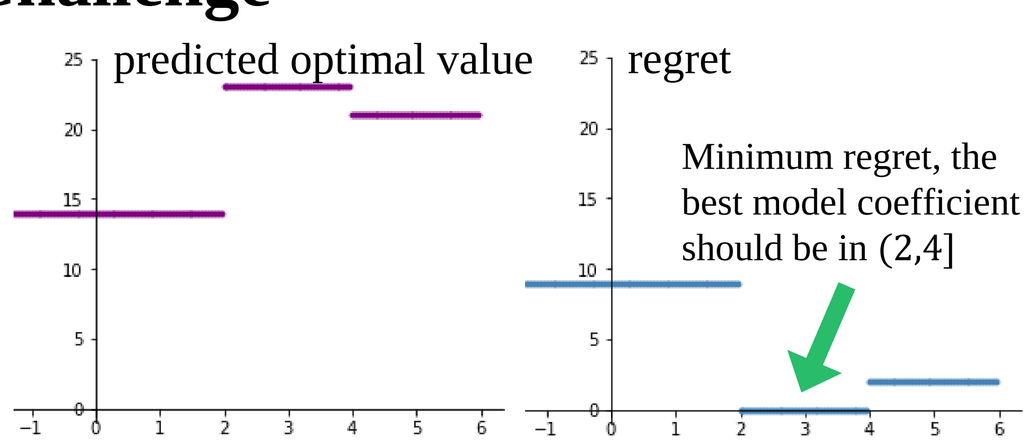


Predict+Optimize
The prediction takes the COP into account.



- Predict+Optimize (P+O) is a kind of approach to solve the COPs with unknown parameters, which takes the COPs into account when doing predictions.
- The metric used in learning of P+O is a new evaluation criterion, called *regret:* compare the difference between the predicted optimal value and the true optimal value

Challenge



The regret function is **piecewise constant** and **non-differentiable**, thus is unfriendly to any gradient-based learning process.

	Approximation Methods	Exact Methods
Core idea	transform the new regret function into a differentiable surrogate function	make use of the problem properties to train the model without computing the gradients
SOTA approach	SPO, QPTL, IntOpt	DNL, SPO Tree, SPO Forest

Contributions

- Limitations of prior works: can deal with only problems solvable with dynamic programming
- Proposed B&L framework: handles recursively and iteratively solvable problems, which is a substantially larger class of optimization problems

Core Idea

- Assumption: the prediction model is linear, $f(A) = A\alpha$ (α: model coefficient vector)
- To train models without computing gradients: adopt the coordinate descent based method proposed by previous work [Demirović et al., AAAI 2020]

The key component of the coordinate descent is the *Convert* function, which is to solve a parameterized COP.

Given a COP solvable by a recursive algorithm satisfying simple conditions, we show how a corresponding parameterized algorithm can be constructed directly and methodically from the recursive solving algorithm.

Algorithm 1: Coordinate Descent
Input: A Para-OP $P(\theta)$ and a training data set $\{(A^1,\theta^1),\ldots,(A^n,\theta^n)\}$
Output: a coefficient vector $\alpha \in \mathbb{R}^m$
Initialize α arbitrarily and $k \leftarrow 0$;
while *not converged* $\wedge$ *resources remain* **do**
...
return α;

Each iteration contains three functions:
Construct()
Convert()
Evaluate()

ReLearn() can be used as Convert() in the coordinate descent algorithm

Function $\texttt{ReSolve}(P,\theta)$**:**
  if $isBaseCase(P)$ **then**
    $(x^*(\theta), C^*(\theta)) \leftarrow \texttt{BaseCase}(P,\theta)$;
  else
    $T \leftarrow \texttt{Extract}(P,\theta)$;
    $PL \leftarrow \texttt{Branch}(P,\theta,T)$;
    $RL \leftarrow \texttt{Map}(\texttt{ReSolve}, PL)$;
    $(x^*(\theta), C^*(\theta)) \leftarrow \texttt{Reduce}(\oplus, RL)$;
  return R;

transform

Function $\texttt{ReLearn}(P_\gamma, I_0)$**:**
  if $isBaseCase(P_\gamma, I_0)$ **then**
    $R[I_0] \leftarrow \texttt{BaseCaseL}(P_\gamma)$;
  else
    $T_\gamma \leftarrow \texttt{ExtractL}(P_\gamma, I_0)$;
    for *each interval* $I \in \mathbb{I}(T_\gamma)$ **do**
      $PL \leftarrow \texttt{BranchL}(P_\gamma, T_\gamma[I])$;
      $RL \leftarrow \texttt{Map}(\texttt{ReLearn}, PL)$;
      $R[I] \leftarrow \texttt{Reduce}(\boxplus, RL)$;
  return R;

- *isBaseCase(), Map(), Reduce()*: unchanged
- *BaseCase(), Extract()*, and *Branch()*: all arithmetic operations replaced by their corresponding piecewise linear generalizations to form *BaseCaseL(), ExtractL()*, and *BranchL()* respectively

Example: Recursive Solving and Recursive Learning for the 0-1 Knapsack Problem

Function $\texttt{ReSolve\_KS}(p,c,n,W,S)$**:**
  if $n = 0$ *or* $W \le 0$ **then**
    $R \leftarrow \mathbb{1}(W \ge 0) \cdot \sum_{i \in S} p[i]$;
  else
    $[P_1, P_2] \leftarrow \texttt{Branch\_KS}(p,c,n,W,S)$;
    $RL \leftarrow \texttt{Map}(\texttt{Resolve\_KS}, PL)$;
    $R \leftarrow \texttt{Reduce}(\max, RL)$;
  return R;

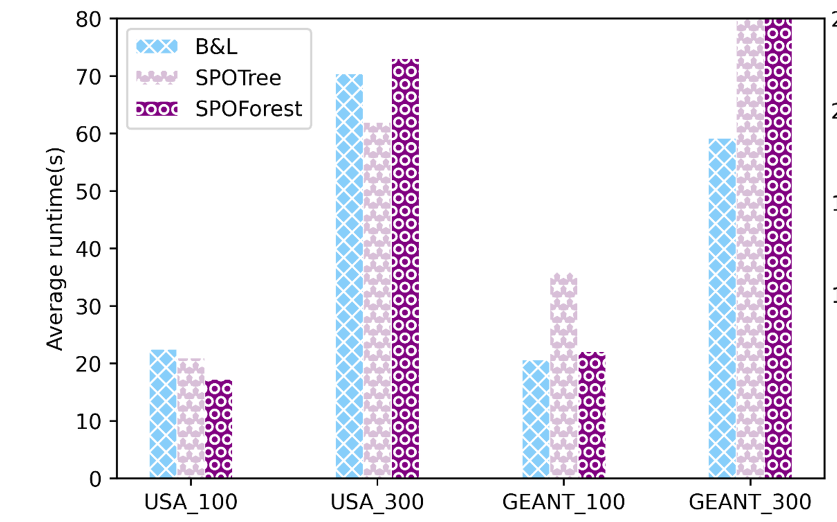
transform

Function $\texttt{ReLearn\_KS}(p_\gamma,c,n,W,S,I)$**:**
  if $n = 0$ *or* $W \le 0$ **then**
    $R[I] \leftarrow \sum_{i \in S} p_\gamma[i]$;
  else
    $[P_1, P_2] \leftarrow \texttt{BranchL\_KS}(p_\gamma,c,n,W,S)$;
    $RL \leftarrow \texttt{Map}(\texttt{ReLearn\_KS}, PL)$;
    $R[I] \leftarrow \texttt{Reduce}(\max, RL)$;
  return $R[I]$;

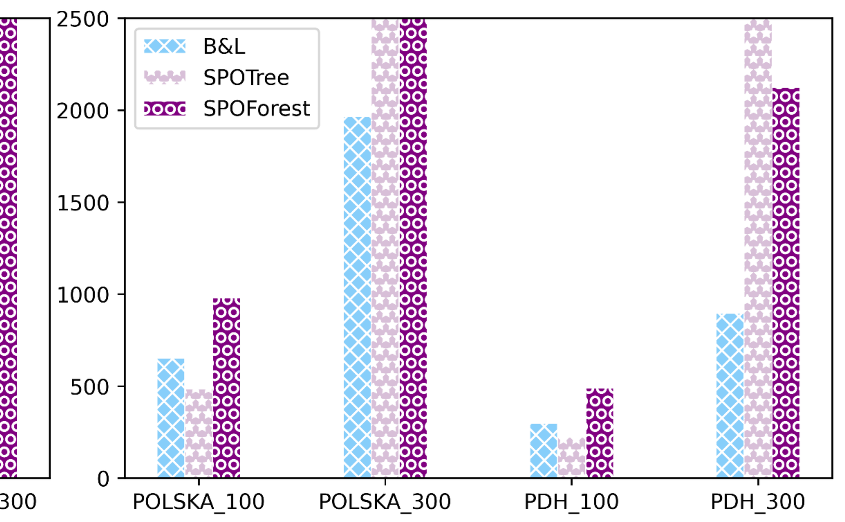
Key Results

Runtime:

- B&L's running time is on par with state-of-the-art approaches (SPO Tree and SPO Forest).
- Consider the ratio of running times between dataset sizes 100 and 300, B&L scales better than SPO Tree and Forest.



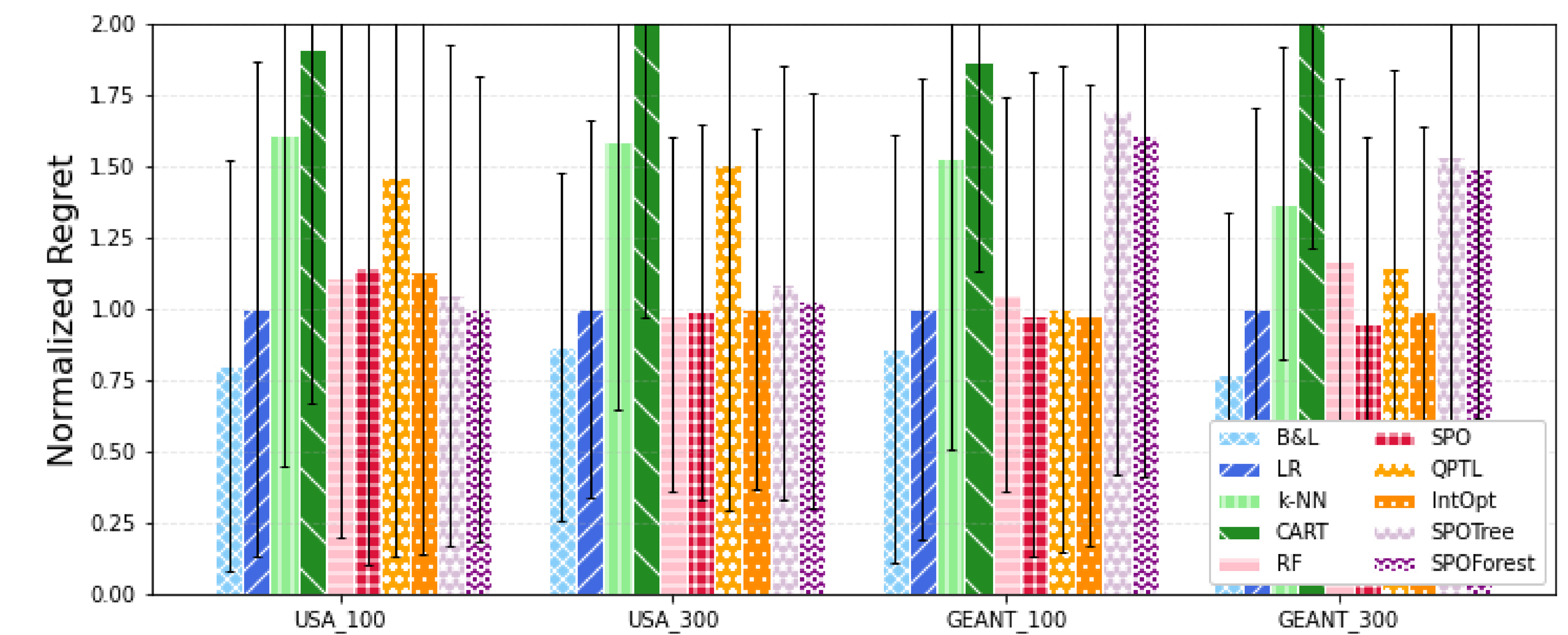
Minimum Cost Flow



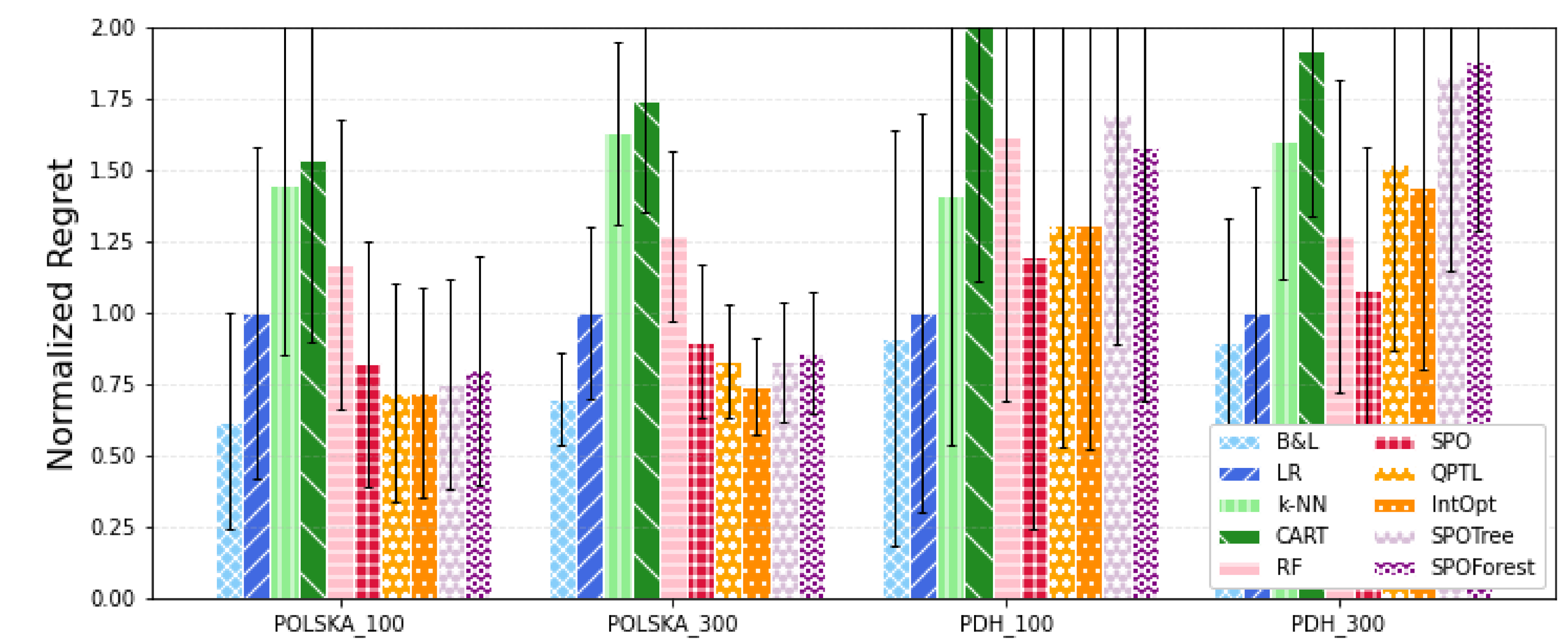
Minimum Cost Vertex Covering

Solution Quality:

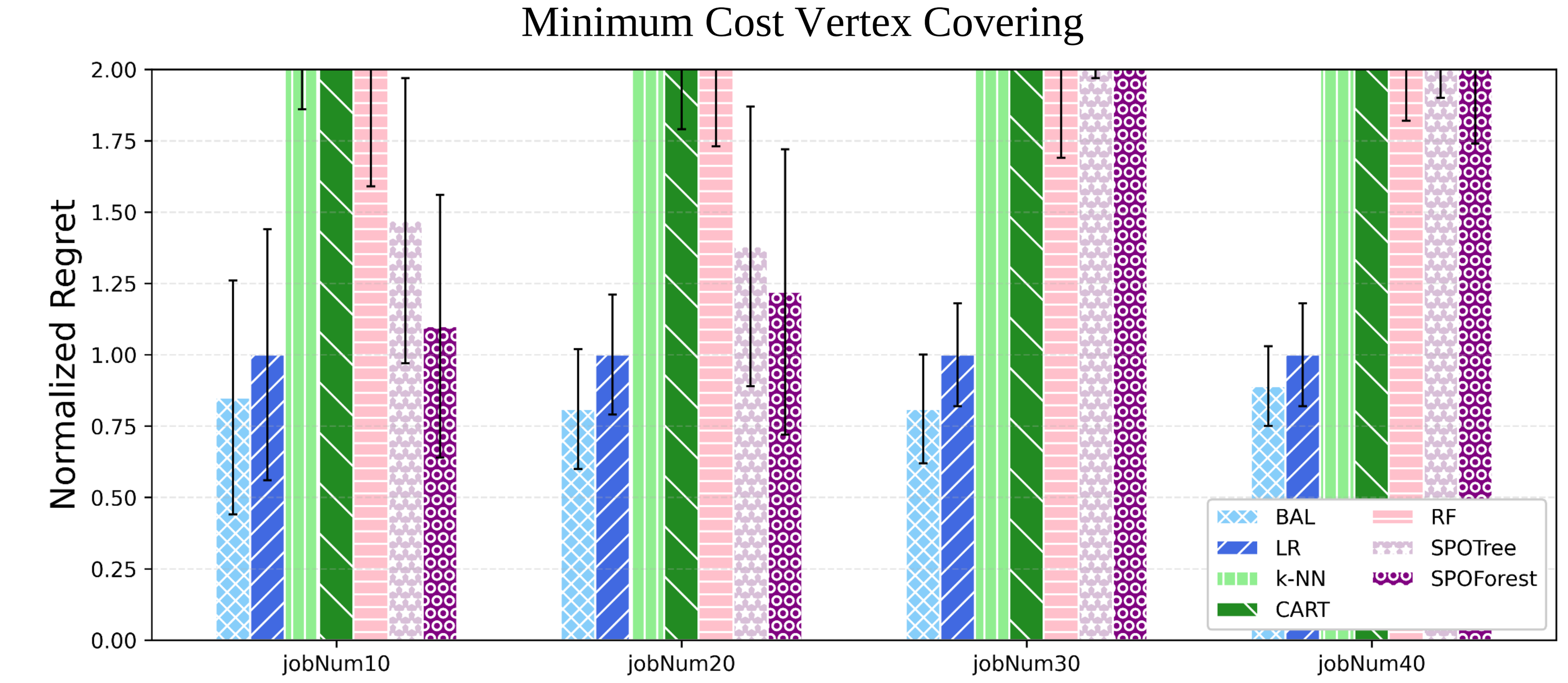
- B&L achieves the best performance over classical and state-of-the-art approaches.



Minimum Cost Flow



Minimum Cost Vertex Covering



Multi-stage Scheduling

Conclusion:
A new framework for Predict+Optimize
- **Handles much larger class of optimization problems**
- **Good empirical performance**