End-to-End PPO-Based Reinforcement Learning Framework for Scalable 0/1 Knapsack Problem Solving

From Data Generation to Large-Scale Generalization

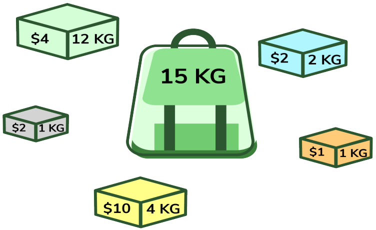
Gang Lin
Student ID: 2874886

University of Birmingham

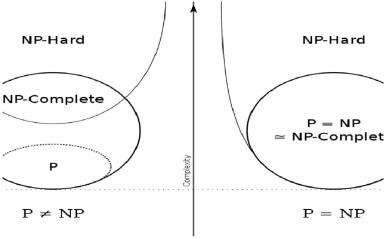
19/08/2025

Contents

The 0/1 Knapsack Problem: Definition & Context



(a) The Knapsack Problem Analogy.



(b) Computational Complexity Classes.

The 0/1 Knapsack Problem

Given n items with weights and values, choose a subset.

- **Objective:** Maximize the total value.
- **Constraint:** The total weight must not exceed the knapsack's capacity.
- **Rule:** Each item is either taken (1) or left (0), no fractions

Applications & Complexity

- **Real-world Applications:** Portfolio selection, resource allocation, logistics.
- **Problem Variants:** Many variants exist by altering constraints or item properties. The 0/1 version is the most common.
- **Computational Hardness:** KP is **NP-complete**. There is no known polynomial-time exact solution.

Chapters

Knapsack Problem Solvers: A Comparative Overview

1. Exact and Approximate Algorithms

Algorithm	**TC.**	**SC.**	**Limitations**
Dynamic Programming	$O(n \cdot C)$	$O(n \cdot C)$	High memory usage; infeasible for large capacity C
Branch & Bound	$O(2^n)$ (worst-case)	$O(n^2)$	Exponential runtime in worst case; performance depends on bounding quality
Greedy	$O(n \log n)$	$O(n)$	No performance guarantee; poor approximation in worst case
Genetic Algorithm	$O(G \cdot P \cdot n)$	$O(P \cdot n)$	Parameter-sensitive; may converge prematurely
Simulated Annealing	$O(\text{iter} \cdot n)$	$O(n)$	Slow convergence; sensitive to cooling schedule

2. Modern Neural Network Approaches (for Knapsack Problem)

Work	**Type & Gen.**	**Architecture / Algorithm**	**Key Results (Accuracy / Speed)**
Bello (2017)	C / Fixed	PtrNet / REINFORCE	Near-optimal on small scale; Faster than exact algorithms
Yildiz (2022)	C / Fixed	Transformer / DQN	$\sim$92.7% accuracy; $\sim$40x faster than DP
Abid (2023)	C / Fixed	MLP / SL	Near-optimal but slower inference than heuristics
Cappart (2021)	I / Yes	DRL + CP Solver	Proves optimality; Outperforms standalone RL/CP

Note: TC. = Time Complexity, SC. = Space Complexity. n: number of items, C: knapsack capacity, G: generations, P: population size, iter: iterations. **C** = Constructive; **I** = Improvement; **Gen.** = Generalization. PtrNet = Pointer Network; MLP = Multi-Layer Perceptron; SL = Supervised Learning; CP = Constraint Programming.

My Contribution: A generalizable RL framework that solves knapsack problems of arbitrary size (train on N, test on $> N$) with 70% accuracy using PPO, enabling scalable and robust decision-making.

Chapters

From Dynamic Programming (DP) to Reinforcement Learning (RL)

1. Dynamic Programming Recurrence for KP

State value is the maximum value by:

- **Skip item i:** Value is the future reward from the remaining state, $V(i-1, w)$.
- **Take item i:** Value is the immediate reward v_i + future reward from the new state, $V(i-1, w-w_i)$.

$$V(i,w) = \begin{cases} V(i-1,w) & \text{if } w_i > w \\ \max(V(i-1,w), \\ v_i + V(i-1, w-w_i)) & \text{if } w_i \leq w \end{cases}$$

2. The Bellman Equation (Element-wise)

State value is the expected immediate reward plus the discounted expected future reward.

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \Big[\sum_{r \in R} p(r|s,a) r + \gamma \sum_{s' \in S} p(s'|s,a) V_\pi(s') \Big]$$

- **DP** recurrence is a specific, deterministic instance of the Bellman equation.

Modeling KP as an RL Problem

- **State (s_t):** The set of available items and the current remaining knapsack capacity.
- **Action (a_t):** The selection of one item from the available set.
- **Reward (R_{t+1}):** The value (v_i) of the selected item.
- **Policy ($\pi_\theta(a|s)$):** A neural network mapping states to action probabilities.

From Bellman to PPO: An Evolutionary Path

1. **Bellman Equation**

$$V^{\pi}(s) = \mathbb{E}[r + \gamma V^{\pi}(s')]$$

2. **Value Function Approx.**

$$V_{\phi}(s) \approx V^{\pi}(s)$$

3. **REINFORCE (Policy Gradient)**

$$\nabla_{\theta} J = \mathbb{E}\left[\nabla_{\theta} \log \pi_{\theta}(a|s) \cdot G_t\right]$$

4. **A2C (Actor-Critic)**

$$\nabla_{\theta} J \propto \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \cdot \hat{A}_t$$

$(\hat{A}_t = r + \gamma V_{\phi}(s') - V_{\phi}(s)$: *TD-based)*

5. **PPO (Clipped Objective)**

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t \hat{A}_t,\ \text{clip}(r_t, 1 \pm \epsilon)\hat{A}_t\right)\right]$$

$$\text{where } r_t = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

The Evolutionary Path to PPO

Bellman → Value Approx. Replace exact V^{π} with a learnable $V_{\phi}(s)$ (e.g., neural net). Enables scalability to large or continuous state spaces.

Value Approx. → REINFORCE Shift from value-based to direct policy optimization. More suitable for stochastic or complex action spaces.

REINFORCE → A2C Replace Monte Carlo return G_t with TD-based advantage $\hat{A}_t$. Reduces variance, enables online updates, and improves sample efficiency.

A2C → PPO Replace policy gradient with clipped surrogate objective. Prevents destructive updates and stabilizes training.

PPO: Algorithm and Architecture

Training Algorithm

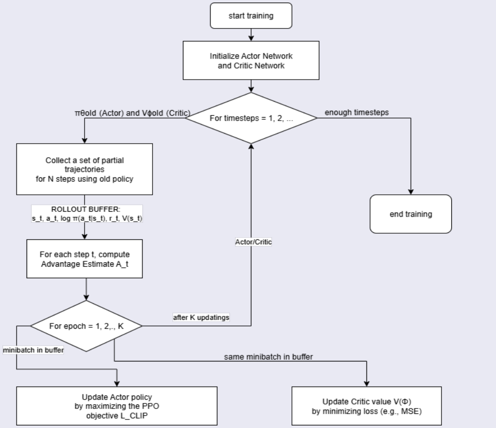


Figure: The PPO training loop using an Actor-Critic framework.

- Multiple optimization on the same minibatch.

Model Architecture

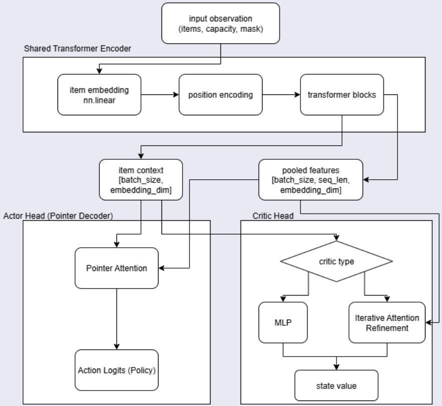


Figure: The model has two heads: one for the policy (Actor) and one for the value (Critic).

- Actor and Critic share the same encoder.

Chapters

Dataset Generation and Preprocessing

Dataset Specification

We generated three distinct datasets for training, validation, and testing to ensure a robust evaluation of the model's generalization capabilities.

Parameter	Training Set	Validation Set	Test Set
Item Count Range (n)	5 to 50	5 to 50	**5 to 200**
Step Size	5	5	5
Instances per Size	100	30	50
Total Instances	**1,000**	**300**	**1,950**

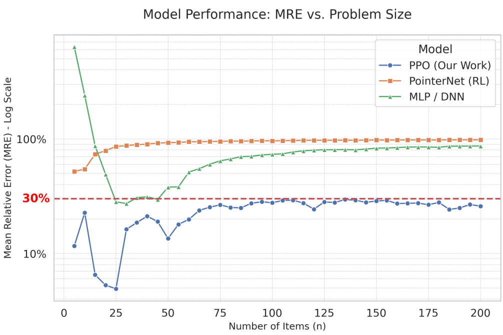
Item Properties

- Weights (w_i) and values (v_i) are integers sampled uniformly from $U[1, 100]$.
- There is no correlation between an item's weight and its value.
- All inputs are normalized before being fed to the model.

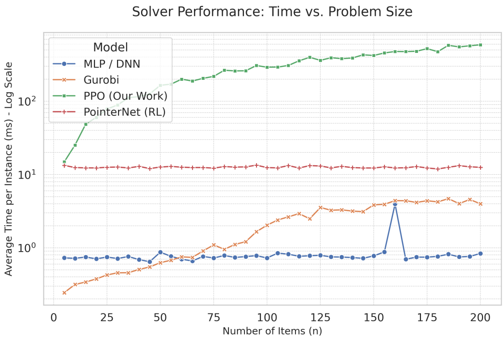
Problem Instance Constraints

- The knapsack capacity (C) is set relative to the total weight of all items ($\sum w_i$).
- The ratio $\frac{C}{\sum w_i}$ is randomly sampled from $U[0.1, 0.9]$.

Results: Accuracy and Inference Time



(a) Mean Relative Error (MRE) vs. Problem Size.



(b) Inference Time vs. Problem Size.

Key Findings: Accuracy

- Our PPO model maintains a low Mean Relative Error (MRE), demonstrating high solution quality and strong generalization.
- Pointer Network shows a higher error rate.
- The pure MLP model fails to generalize effectively.

Key Findings: Inference Time

- PPO's inference time is practical for large instances.
- Pointer Network is faster but less accurate.
- MLP is the fastest but provides poor solutions.

Conclusion

Performance Summary

- **PPO vs. Pointer Network (Accuracy):**
  - **Algorithmic Superiority:** PPO's Actor-Critic (TD) method provides low-variance updates.
  - **Architectural Advantage:** The **Transformer** encoder captures the global, combinatorial nature of the problem more effectively than a sequential **LSTM**.
  - **Framework Robustness:** Leveraging **Stable Baselines 3** provides key stabilizations like adaptive observation normalization ('VecNormalize').
- **PPO vs. Pointer Network (Speed):**
  - **Core Architecture:** **Transformer** is more computationally intensive than **LSTM**.
  - **Model Components:** Extra Critic Network requires extra computation.
  - **Evaluation Method:** Stable_baseline3 cannot support batch evaluation.

Effective Training Techniques

The success of the framework relies on several key techniques:

- **Input Normalization:**
  Normalizing item attributes (w_i, v_i) and the knapsack capacity (C) is crucial.
- **Observation & Reward Normalization:**
  Using 'VecNormalize' for both observations and rewards stabilizes the learning process significantly.
- **Heuristic Preprocessing:**
  Sorting items by value-density (v_i/w_i) before feeding them to the model provides a strong inductive bias and improves performance.

Future Work & Open Questions

Architectural Exploration

- **The "Simple Critic" Anomaly:**
  A simple MLP Critic achieved higher accuracy (70%) than a more complex attention-based head (60%). Future work should investigate if this is due to optimization challenges or a regularization effect.
- **Global State Representation (‘[CLS]‘ Token):**
  Initial experiments with a ‘[CLS]‘ token for global state representation surprisingly decreased performance. This warrants further investigation.
- **Hyperparameter Tuning:**
  While a 3-layer MLP Critic works well, its optimal width and the interplay with network depth remain open questions for further tuning.

Problem Formulation & Reward Shaping

- **Explore Alternative Formulation:** Our model uses a **"Decision" formulation** (select one from all remaining items). An alternative **"Selection" formulation** (decide 'take' or 'skip' for items sequentially) could be investigated.
- **Advanced Reward Shaping:** For the current "Decision" model, an initial attempt at adding a final shaping reward (to encourage a fuller knapsack) decreased accuracy. Further research into more advanced shaping techniques (e.g., potential-based rewards) is needed.

Thank You Q & A