We want to design algorithms that satisfy the following key properties.

- **Consistency**: Performance of the algorithm should improve if given good predictions.
- **Robustness**: Performance of the algorithm should degrade gracefully in case of bad predictions.
- **Independence**: The algorithm should be independent of the predictor. In particular, it should make no assumptions on the types / distribution of the predictor errors.

### Competitive Ratio

\[
\text{Competitive Ratio} = \max_{x} \frac{\text{Cost}(\text{Algorithm})}{\text{Cost}(\text{Optimum})}.
\]

- For online algorithms with predictions, this is a function \(c(\eta)\) of the error \(\eta\) of the prediction.
- Algorithm is \(\beta\)-consistent: \(c(0) = \beta\).
- Algorithm is \(\gamma\)-robust: \(c(\eta) \leq \gamma, \forall \eta\)

### Related Work

- Improve reserve price optimization via prediction oracles (Medina and Vassilvitskii, 2017)
- Online Caching with Predictions (Lykouris and Vassilvitskii, 2018)

---

**Ski Rental**

**Problem Definition**: A skier wishes to ski for an unknown number \(x\) of days. She can either rent skis for \$1 each day, or buy skis for \$6 and ski for free thereafter. When should she buy the skis?

**Prediction**: \(p\) — number of days the skier is predicted to ski.

For example, one can build a model using
- Weather forecasts
- Location, etc.

### Naive Algorithm

- Trust the prediction completely.
  - Consistency: 1
  - Robustness: \(\infty\)

\[
\text{if } y \geq b \text{ then } \begin{cases} 
\text{Buy on the first day.} \\
\text{Rent every day.} 
\end{cases}
\]

### Consistent and Robust Algorithms

- Hedge your bets!
  - Tradeoff consistency vs. robustness via \(\lambda\)
  - Consistency: \((1 + \lambda)\)
  - Robustness: \(1 + \frac{1}{\lambda}\)

\[
\text{if } y \geq b \text{ then } \begin{cases} 
\text{Buy on day } \lfloor b/\lambda \rfloor \\
\text{Buy on day } \lceil b/\lambda \rceil 
\end{cases}
\]

### Main Theorem: Ski Rental

For any \(\lambda \in (0, 1)\), there is a randomized online algorithm for Ski-Rental that is \(\frac{1}{1 - e^{-\lambda}}\) robust and \(\frac{\lambda}{1 - e^{-\lambda}}\) consistent.

**Using Predictions**

**Hedge your bets**: Preferential Round-Robin (PRR): Run SPJF at \(\lambda\) capacity and RR at \((1 - \lambda)\) capacity

- Consistency: \(\frac{\lambda}{1 - \lambda}\)
  - Robustness: \(\frac{\lambda}{1 - \lambda}\)

---

**Non-Clairvoyant Scheduling**

**Problem Definition**: Schedule jobs of unknown length on one machine to minimize the sum of completion times. Preemption is allowed.

**Prediction**: Predicted length of each job. Could be from a model based on lengths of previous runs of the same job or on past jobs with similar parameters.

**Best Algorithms without Predictions**

- **Clairvoyant**: Shortest Job First ( SJF ): optimal
- **Non-clairvoyant**: Round-Robin ( RR ): 2-competitive in the worst case

### Using Predictions

**CLAIRVOYANT**

- **Consistency**: 1
  - Robustness: \(\infty\)

**Non-CLAIRVOYANT**

- **Consistency**: 2
  - Robustness: 2

**Hedge your bets**: Preferential Round-Robin (PRR): Run SPJF at \(\lambda\) capacity and RR at \((1 - \lambda)\) capacity.

**Using Predictions**

- Consistency: \(\frac{\lambda}{1 - \lambda}\)
  - Robustness: \(\frac{\lambda}{1 - \lambda}\)

**Main Theorem: Scheduling**

PFR algorithm with \(\lambda \in (0, 1)\) has competitive ratio \(\min\left\{ \frac{1}{1 + \sqrt{\lambda}}, \frac{1}{1 - \sqrt{\lambda}} \right\}\).