Data driven techniques for learning games playing strategies and their application to combinatorial optimization problems

Thomas Philip Runarsson

School of Engineering and Natural Sciences
University of Iceland

Stirling University 6 Dec. 2013

Outline

Sequential Decision Making Problems

Many optimization problems may be formulated within a sequential decision making (dynamic programming) framework, this includes the shortest path, assignment, packing, scheduling, etc.

A solution consist of n components, or decisions selected one-at-a-time. For $n = 1, \ldots, N$, the state of the nth stage is formed by the sequence of n decisions:

$$(u_1, u_2, \ldots, u_n)$$

For example, in *job scheduling* decisions may involve selecting different dispatching heuristic or simply the job to be dispatched next.

Sequential Decision Making Problems

Many optimization problems may be formulated within a sequential decision making (dynamic programming) framework, this includes the shortest path, assignment, packing, scheduling, etc.

A solution consist of n components, or decisions selected one-at-a-time. For $n = 1, \dots, N$, the state of the nth stage is formed by the sequence of n decisions:

$$(u_1, u_2, \dots, u_n)$$

For example, in *job scheduling* decisions may involve selecting different dispatching heuristic or simply the job to be dispatched next.

Sequential Decision Making Problems

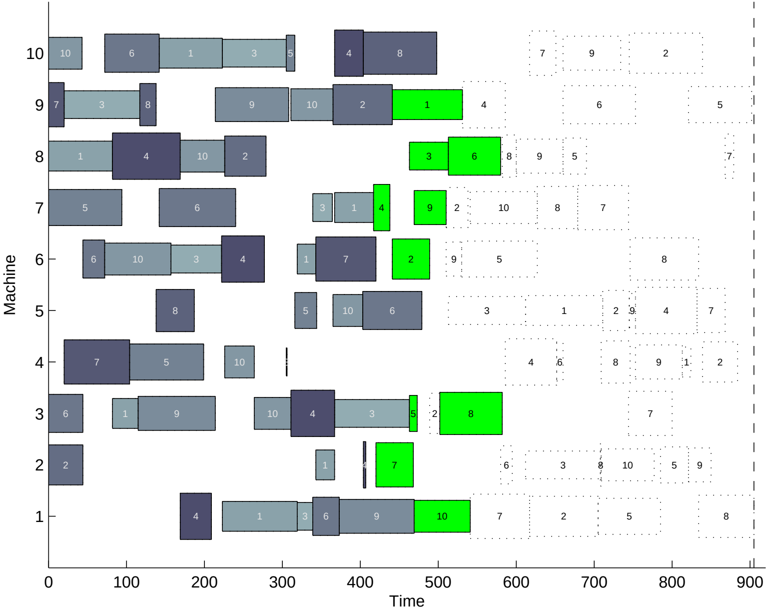
Many optimization problems may be formulated within a sequential decision making (dynamic programming) framework, this includes the shortest path, assignment, packing, scheduling, etc.

A solution consist of n components, or decisions selected one-at-a-time. For $n = 1, \ldots, N$, the state of the nth stage is formed by the sequence of n decisions:

$$(u_1, u_2, \ldots, u_n)$$

For example, in *job scheduling* decisions may involve selecting different dispatching heuristic or simply the job to be dispatched next.

Jobshop problem as an example ...



Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The key idea is to employ a given heuristic in the construction of an optimal cost-to-go function approximation, which is then used in the spirit of the *neuro-dynamic programming* and *reinforcement learning* methodology.

In particular, an optimal solution $(u_1^*, \ldots, u_N^*)$ can be obtained by

$$u_i^* = \arg \min_{u_i \in U_i(u_1^*, \ldots, u_{i-1}^*)} J^*(u_1^*, \ldots, u_{i-1}^*, u_i), \quad i = 1, \ldots, N$$

The Rollout algorithm uses multiple *heuristics* to provide an approximation of J^* and so obtain a sub-optimal solution

$$\tilde{u}_i = \arg \min_{\tilde{u}_i \in U_i(\tilde{u}_1, \ldots, \tilde{u}_{i-1})} \tilde{J}(\tilde{u}_1, \ldots, \tilde{u}_{i-1}, u_i), \quad i = 1, \ldots, N$$

Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The key idea is to employ a given heuristic in the construction of an optimal cost-to-go function approximation, which is then used in the spirit of the *neuro-dynamic programming* and *reinforcement learning* methodology.

In particular, an optimal solution $(u_1^*, \ldots, u_N^*)$ can be obtained by

$$u_i^* = \arg \min_{u_i \in U_i(u_1^*, \ldots, u_{i-1}^*)} J^*(u_1^*, \ldots, u_{i-1}^*, u_i), \quad i = 1, \ldots, N$$

The Rollout algorithm uses multiple *heuristics* to provide an approximation of J^* and so obtain a sub-optimal solution

$$\tilde{u}_i = \arg \min_{\tilde{u}_i \in U_i(\tilde{u}_1, \ldots, \tilde{u}_{i-1})} \tilde{J}(\tilde{u}_1, \ldots, \tilde{u}_{i-1}, u_i), \quad i = 1, \ldots, N$$

Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The key idea is to employ a given heuristic in the construction of an optimal cost-to-go function approximation, which is then used in the spirit of the *neuro-dynamic programming* and *reinforcement learning* methodology.

In particular, an optimal solution $(u_1^*, \dots, u_N^*)$ can be obtained by

$$u_i^* = \arg \min_{u_i \in U_i(u_1^*, \dots, u_{i-1}^*)} J^*(u_1^*, \dots, u_{i-1}^*, u_i), \quad i = 1, \dots, N$$

The Rollout algorithm uses multiple *heuristics* to provide an approximation of J^* and so obtain a sub-optimal solution

$$\tilde{u}_i = \arg \min_{\tilde{u}_i \in U_i(\tilde{u}_1, \dots, \tilde{u}_{i-1})} \tilde{J}(\tilde{u}_1, \dots, \tilde{u}_{i-1}, u_i), \quad i = 1, \dots, N$$

Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The name “rollout policy”was used by Tesauro (Tesauro and Galperin, 1996) in connection with one of his simulation-based computer backgammon algorithms, also known as *trajectory sampling* (Sutton and Barto, 1998)

There are different versions of the Rollout algorithm, one in particular looks at all downstream neighbour states, $\mathcal{N}(u_{i-1})$, of the partial solution $(\tilde{u}_1, \ldots, \tilde{u}_{i-1})$: and uses a default heuristic to generate a complete solution with cost $C(j)$.

The decision made is then

$$\tilde{u}_i = \arg \min_{j \in \mathcal{N}(\tilde{u}_{i-1})} C(j)$$

Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The name “rollout policy”was used by Tesauro (Tesauro and Galperin, 1996) in connection with one of his simulation-based computer backgammon algorithms, also known as *trajectory sampling* (Sutton and Barto, 1998)

There are different versions of the Rollout algorithm, one in particular looks at all downstream neighbour states, $\mathcal{N}(u_{i-1})$, of the partial solution $(\tilde{u}_1, \ldots, \tilde{u}_{i-1})$: and uses a default heuristic to generate a complete solution with cost $C(j)$.

The decision made is then

$$\tilde{u}_i = \arg \min_{j \in \mathcal{N}(\tilde{u}_{i-1})} C(j)$$

Rollout Algorithms (Bertsekas, Tsitsiklis, 1997)

The name "rollout policy"was used by Tesauro (Tesauro and Galperin, 1996) in connection with one of his simulation-based computer backgammon algorithms, also known as *trajectory sampling* (Sutton and Barto, 1998)

There are different versions of the Rollout algorithm, one in particular looks at all downstream neighbour states, $\mathcal{N}(u_{i-1})$, of the partial solution $(\tilde{u}_1, \ldots, \tilde{u}_{i-1})$: and uses a default heuristic to generate a complete solution with cost $C(j)$.

The decision made is then

$$\tilde{u}_i = \arg \min_{j \in \mathcal{N}(\tilde{u}_{i-1})} C(j)$$

Pilot Method (Duin and Voß, 1994.)

Preferred Iterative LOok ahead Technique.

- The idea is to add one-step look-ahead and so apply greedy heuristics from different starting points.
- The procedure is applied repeatedly, effectively building a tree.
- This procedure is not unlike strategies used in game playing programs, that search a game trees for good moves.
- Essentially equivalient to the Rollout algorithm, but motivated differently.

Pilot Method (Duin and Voß, 1994.)

Preferred Iterative LOok ahead Technique.

- The idea is to add one-step look-ahead and so apply greedy heuristics from different starting points.
- The procedure is applied repeatedly, effectively building a tree.
- This procedure is not unlike strategies used in game playing programs, that search a game trees for good moves.
- Essentially equivalient to the Rollout algorithm, but motivated differently.

Pilot Method (Duin and Voß, 1994.)

Preferred Iterative LOok ahead Technique.

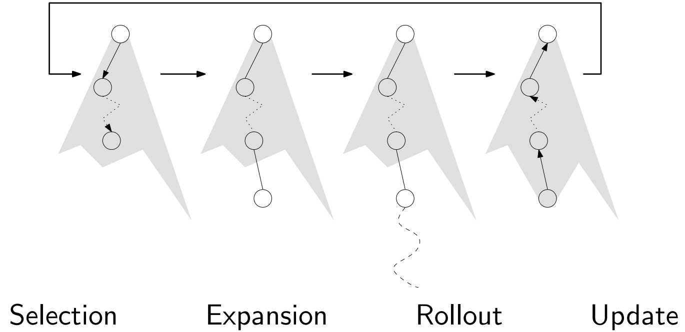
- The idea is to add one-step look-ahead and so apply greedy heuristics from different starting points.
- The procedure is applied repeatedly, effectively building a tree.
- This procedure is not unlike strategies used in game playing programs, that search a game trees for good moves.
- Essentially equivalient to the Rollout algorithm, but motivated differently.

Pilot Method (Duin and Voß, 1994.)

Preferred Iterative LOok ahead Technique.

- The idea is to add one-step look-ahead and so apply greedy heuristics from different starting points.
- The procedure is applied repeatedly, effectively building a tree.
- This procedure is not unlike strategies used in game playing programs, that search a game trees for good moves.
- Essentially equivalient to the Rollout algorithm, but motivated differently.

Monte-Carlo Tree Search



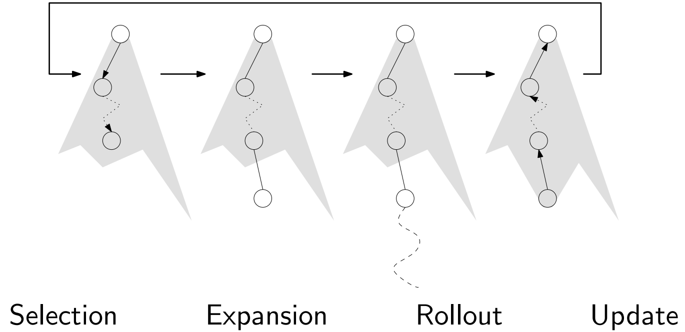
Selection A tree is asymmetrically grown toward the most promising region.

Expansion Tree descended using an exploration/exploitation policy until a unexplored leaf found.

Rollout Node added to the tree and solution completed by some procedure.

Update Solution back-propagated to nodes on the path taken.

Monte-Carlo Tree Search



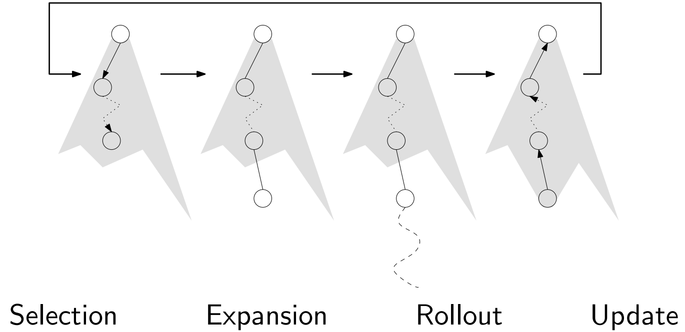
Selection A tree is asymmetrically grown toward the most promising region.

Expansion Tree descended using an exploration/exploitation policy until a unexplored leaf found.

Rollout Node added to the tree and solution completed by some procedure.

Update Solution back-propagated to nodes on the path taken.

Monte-Carlo Tree Search



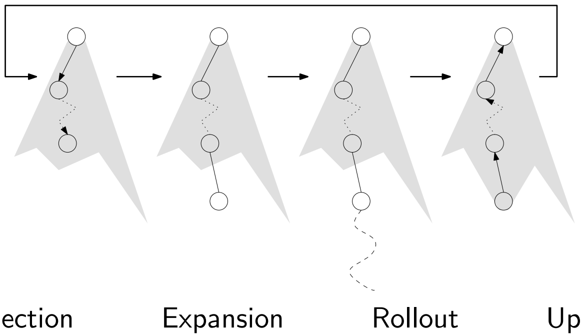
Selection A tree is asymmetrically grown toward the most promising region.

Expansion Tree descended using an exploration/exploitation policy until a unexplored leaf found.

Rollout Node added to the tree and solution completed by some procedure.

Update Solution back-propagated to nodes on the path taken.

Monte-Carlo Tree Search



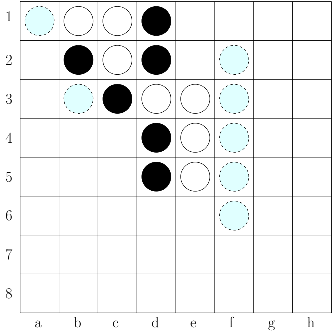
Selection A tree is asymmetrically grown toward the most promising region.

Expansion Tree descended using an exploration/exploitation policy until a unexplored leaf found.

Rollout Node added to the tree and solution completed by some procedure.

Update Solution back-propagated to nodes on the path taken.

Decision Making in Board Games

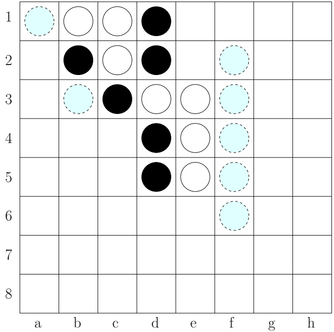


Othello game in progress with seven possible legal moves for black (dash).

The purpose of learning a board evaluation function is to decide which move to take.

- When used in a one-ply search or a minimax game tree, decisions are based on comparisons.
- The absolute values are not important, only relative values are needed.

Decision Making in Board Games

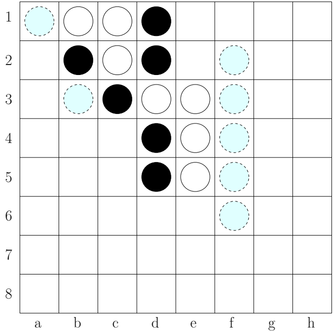


Othello game in progress with seven possible legal moves for black (dash).

The purpose of learning a board evaluation function is to decide which move to take.

- When used in a one-ply search or a minimax game tree, decisions are based on comparisons.
- The absolute values are not important, only relative values are needed.

Decision Making in Board Games



Othello game in progress with seven possible legal moves for black (dash).

The purpose of learning a board evaluation function is to decide which move to take.

- When used in a one-ply search or a minimax game tree, decisions are based on comparisons.
- The absolute values are not important, only relative values are needed.

Preference Learning

- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

Preference Learning

- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

Preference Learning

- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

Preference Learning

- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

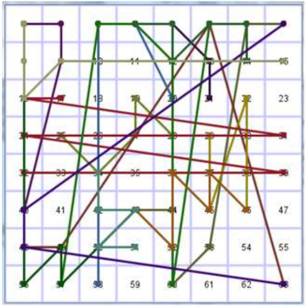
Preference Learning

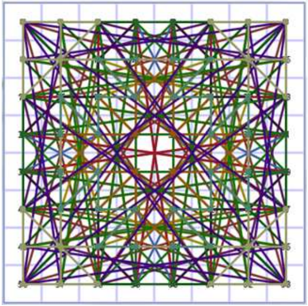
- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

Preference Learning

- Simply learn to satisfy constraints that lead to correct choices.
- Don't care about absolute values.
- What are correct choices?
- Given a set of game logs (trajectories)
- For each board state with more than one legal move:
  - label board state reached by chosen move as "correct".
  - label all other board states reachable by a single legal move as "incorrect".
- Attempt to learn a function that performs correct classification according to the above.

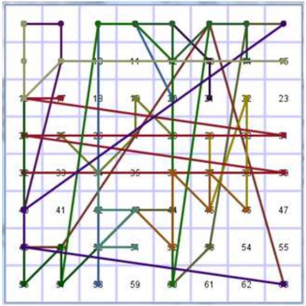
A linear evaluation function based on *n*-tuples features

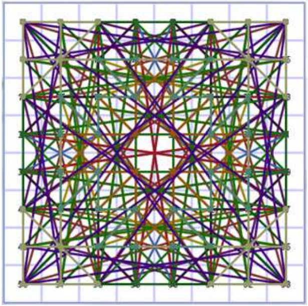




- Essentially a linear architecture with highly non-linear pattern-like features ϕ.
- In our study we have used an *n*-tuple architecture evolved by Pete Burrow, with 6561 features.
- The evaluation function is simply $Q(s, a) = \mathbf{w}^\top \phi(s^a)$, where s^a is the after or post-decision state when move a is chosen.

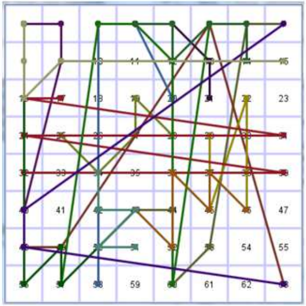
A linear evaluation function based on n-tuples features

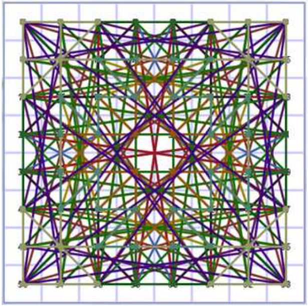




- Essentially a linear architecture with highly non-linear pattern-like features ϕ.
- In our study we have used an n-tuple architecture evolved by Pete Burrow, with 6561 features.
- The evaluation function is simply $Q(s, a) = \mathbf{w}^{\top}\phi(s^{a})$, where s^{a} is the after or post-decision state when move a is chosen.

A linear evaluation function based on n-tuples features





- Essentially a linear architecture with highly non-linear pattern-like features ϕ.
- In our study we have used an n-tuple architecture evolved by Pete Burrow, with 6561 features.
- The evaluation function is simply $Q(s,a) = \mathbf{w}^\top \phi(s^a)$, where s^a is the after or post-decision state when move a is chosen.

Preference Learning

Say that move j is preferred to move k then the learner simply aims to satisfy the simple constraint:

$$\mathbf{w}^\intercal \phi_j > \mathbf{w}^\intercal \phi_k$$

or solve the problem of minimizing $\|\mathbf{w}\|$ and satisfying

$$[\mathbf{w}^\intercal (\phi_j - \phi_k)] > 1 \;\forall j \in J,\; k \in K$$

where J and K are the preferred and non-preferred post-decision board states respectively.

Other Machine Learning Approaches

We (with Simon Lucas) have compared preference learning with the following approaches (and variations thereof):

- least squares temporal difference learning,
- direct classification,
- and the Bradley-Terry model fitted using minorization-maximization.

Human Generated Expert Games Trajectories

- Taken from human competitions held by the French Othello Federation www.ffothello.org.
- More than 112 thousand games available, we use only one thousand.

Matching human decisions (French Othello league)

#discs	BF	#N	*n*-Tuple PREF	MM	LSTD	Classify	WPC	ETDL
1–16	7.1	73133	78.3	68.8	29.8	68.7	21.9	13.4
17–20	11.0	40045	52.4	43.0	15.2	31.8	2.6	15.5
21–24	11.5	42194	49.6	33.4	21.6	32.2	2.0	20.6
25–28	11.9	43796	45.1	31.1	20.7	34.1	5.2	22.9
29–32	11.7	42818	40.5	26.2	18.4	29.7	4.3	22.1
33–36	11.3	41319	40.1	28.0	17.6	30.2	6.8	24.9
37–40	10.6	38318	41.8	30.6	17.9	32.0	9.4	26.1
41–44	9.6	34308	41.5	31.6	20.4	32.5	14.3	29.6
45–48	8.4	29412	43.6	34.0	21.6	34.7	20.8	31.5
49–52	7.1	23784	44.0	35.3	24.1	36.4	27.2	35.8
53–56	5.5	17385	49.0	40.9	31.3	41.1	33.4	42.2
57–60	4.0	10960	53.9	46.5	39.0	48.3	38.7	49.5
61–64	2.5	3411	62.5	55.9	52.8	57.3	48.0	61.3
$\sum$	8.6	(437883)	53.0	42.5	25.1	42.7	17.6	27.0

Round Robin League

- Each evaluation function was used to play each other one using one-ply minimax search from the same 1000 randomly chosen unique initial positions.
- We then used BayesElo to rank the players and to assess the likelihood of superiority.

Round Robin League

- Each evaluation function was used to play each other one using one-ply minimax search from the same 1000 randomly chosen unique initial positions.
- We then used BayesElo to rank the players and to assess the likelihood of superiority.

Round Robin League Rating

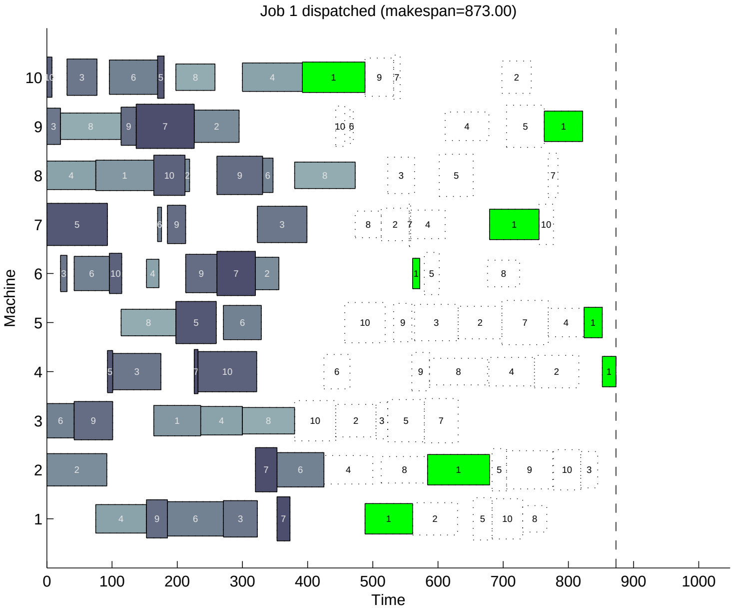
Ranking	Player	Rating	% Score	b/w	$\lvert w \rvert$
1	iPref-N-Tuple	1879	82.3%	inv	6,561
2	ETDL-N-Tuple	1871	81.6%	neg	6,561
3	Pref-N-Tuple	1779	72.3%	neg	6,561
4	Coev-WPC	1672	59.7%	neg	64
5	Heur-WPC	1655	57.5%	neg	64
6	MM-N-Tuple	1630	54.3%	inv	6,561
7	iPref-1-Tuple	1555	44.7%	inv	192
8	MM-1-Tuple	1542	43.0%	inv	192
9	Pref-1-Tuple	1511	39.1%	neg	192
10	Classify-N-Tuple	1500	37.7%	inv	6,561
11	Classify-1-Tuple	1425	28.5%	inv	192
12	LSTD-N-Tuple	1419	27.8%	neg	6,561
13	LSTD-1-Tuple	1360	21.6%	neg	192

Data Driven Design of Composite Dispatching Rules

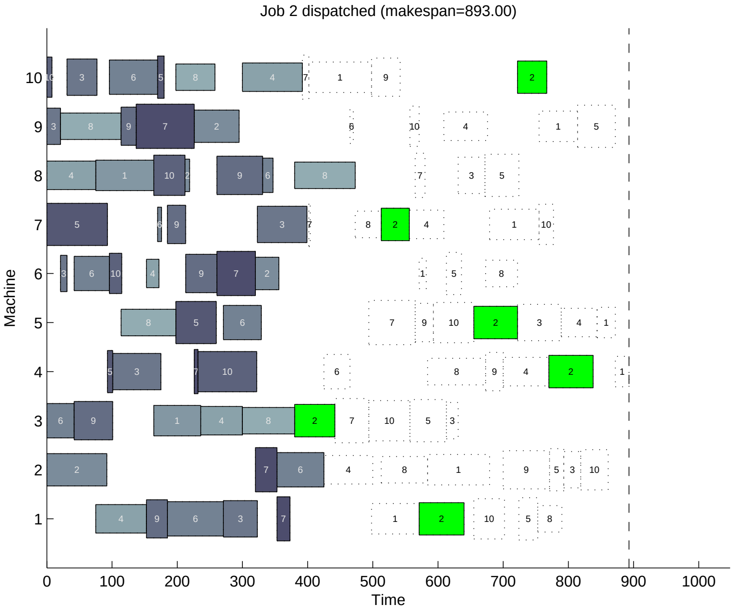
Use now the same preference learning technique for learning dispatching policies for scheduling problems:

- Generate optimal dispatching "trajectories" using a MIP solver (Gurobi).
- Use random instance generator to create example problems to train on and others for testing.
- We will look at 3 different types of scheduling problems.

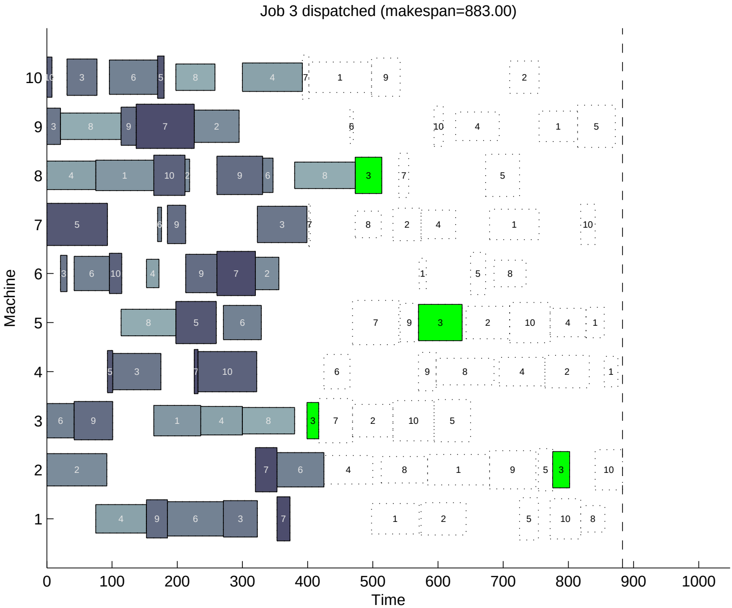
Jobshop processing times $U(1, 100)$ order random



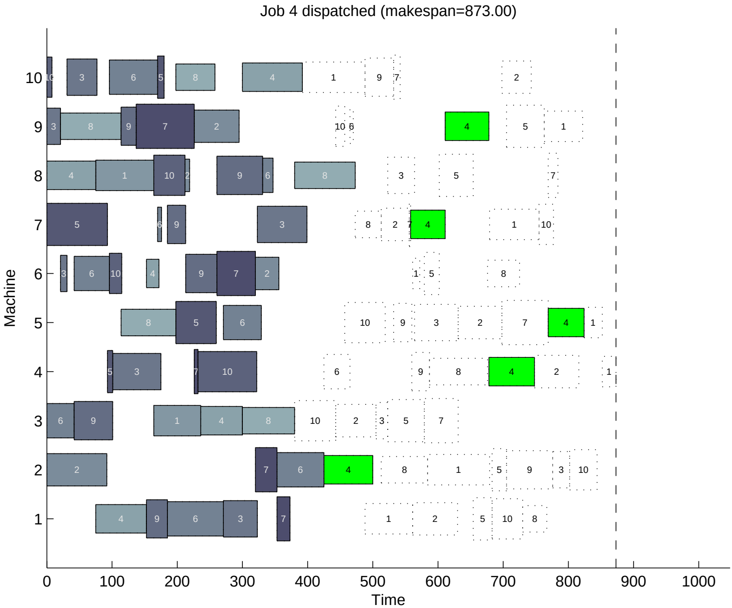
Jobshop processing times $U(1, 100)$ order random



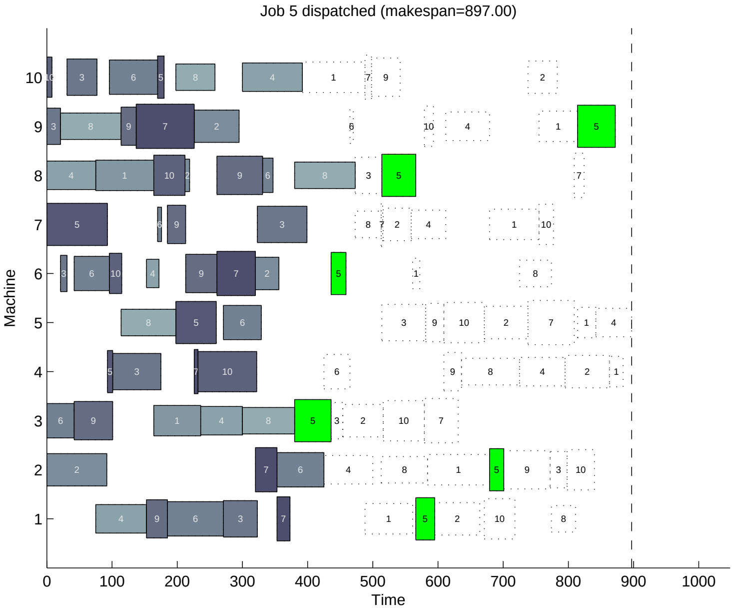
Jobshop processing times $U(1, 100)$ order random



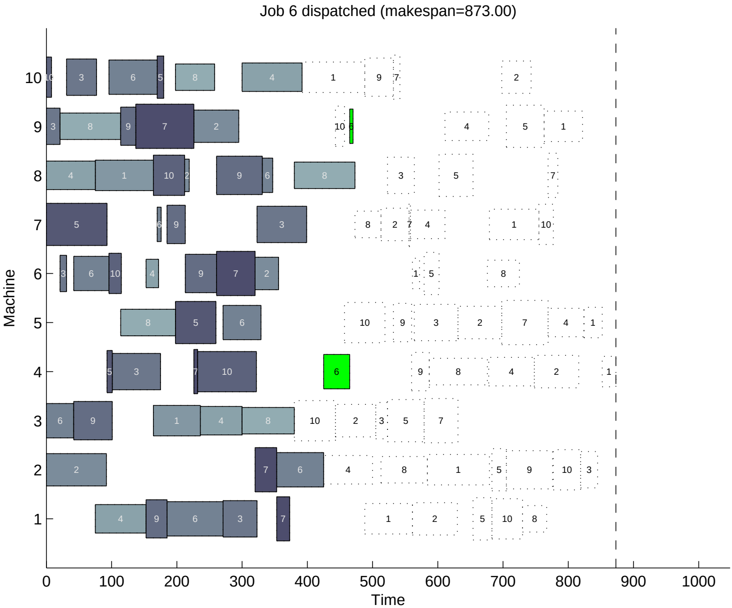
Jobshop processing times $U(1, 100)$ order random



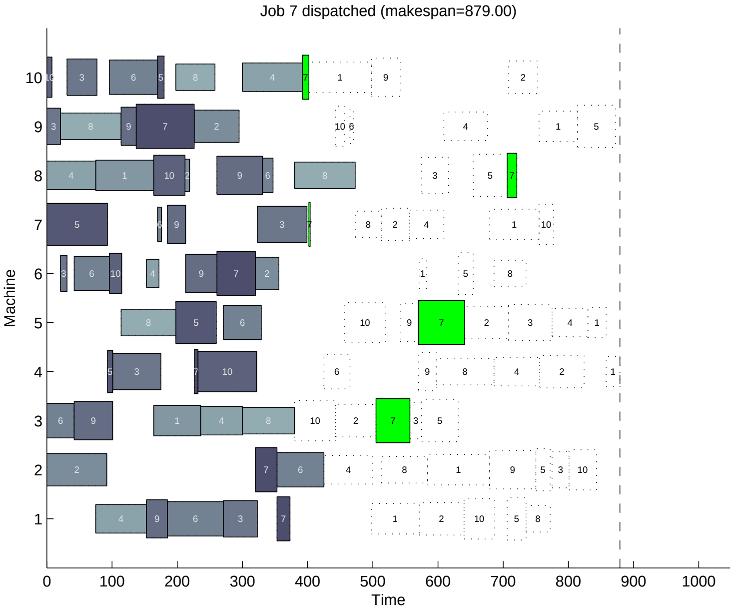
Jobshop processing times $U(1, 100)$ order random



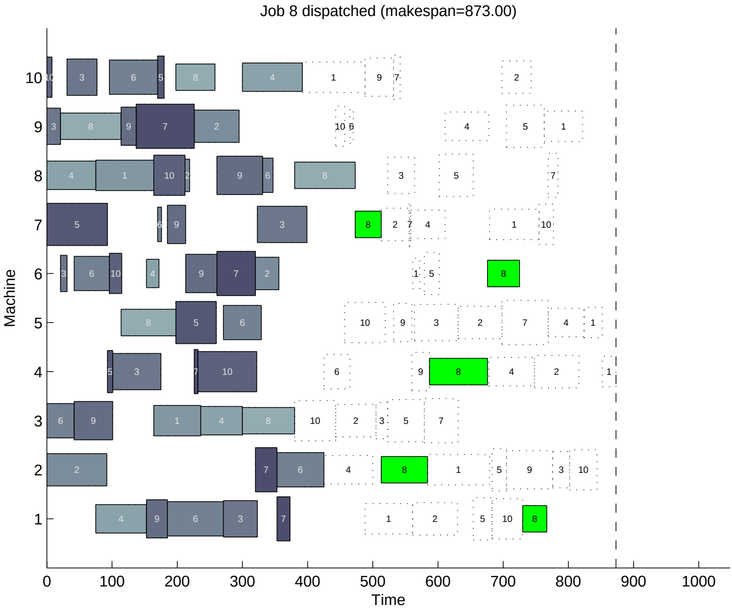
Jobshop processing times $U(1, 100)$ order random



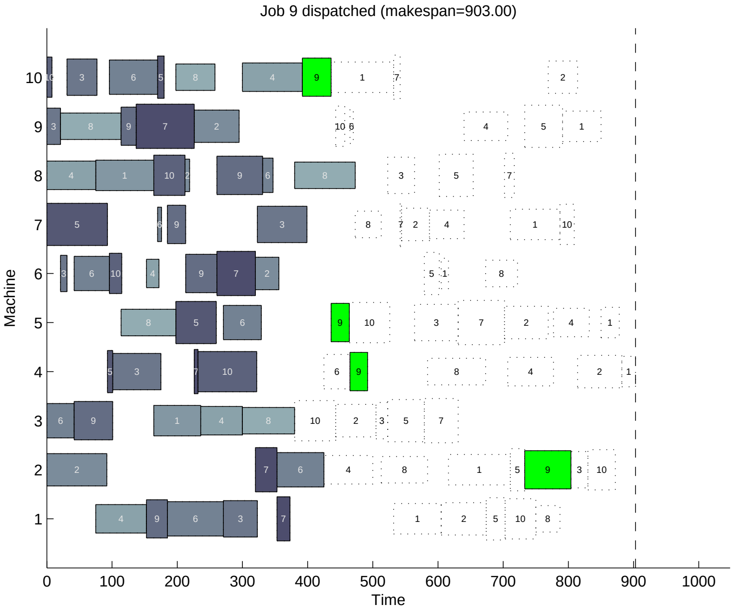
Jobshop processing times $U(1, 100)$ order random



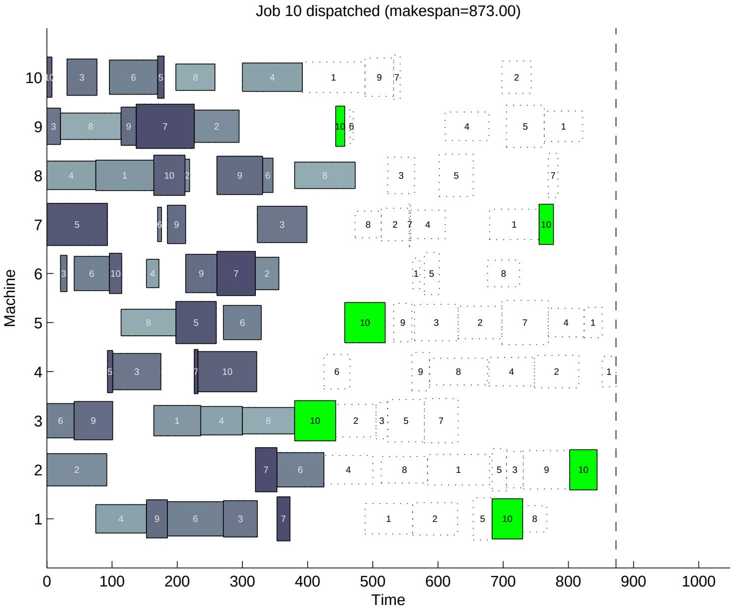
Jobshop processing times $U(1, 100)$ order random



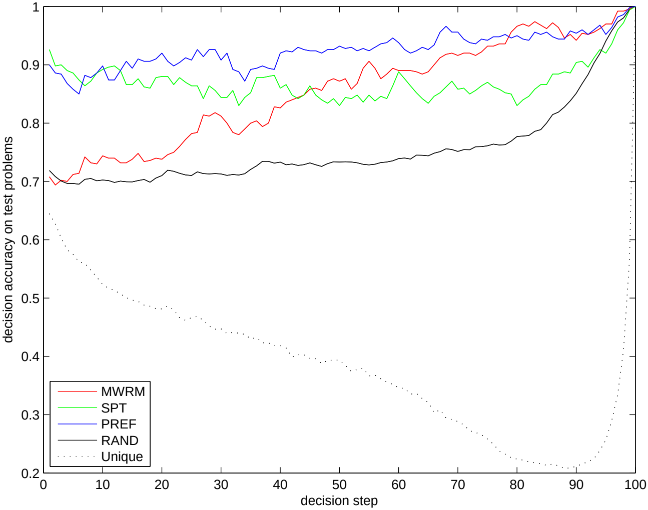
Jobshop processing times $U(1, 100)$ order random



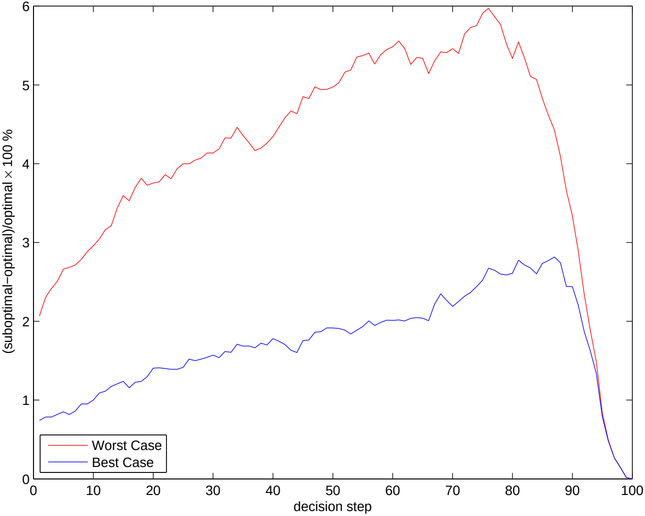
Jobshop processing times $U(1, 100)$ order random



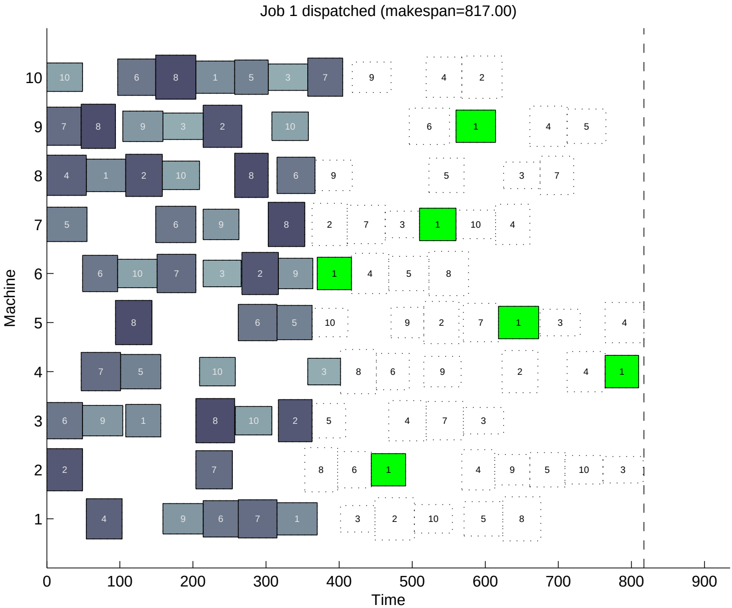
Jobshop-10×10, $U(1, 100)$ – average decision accuracy



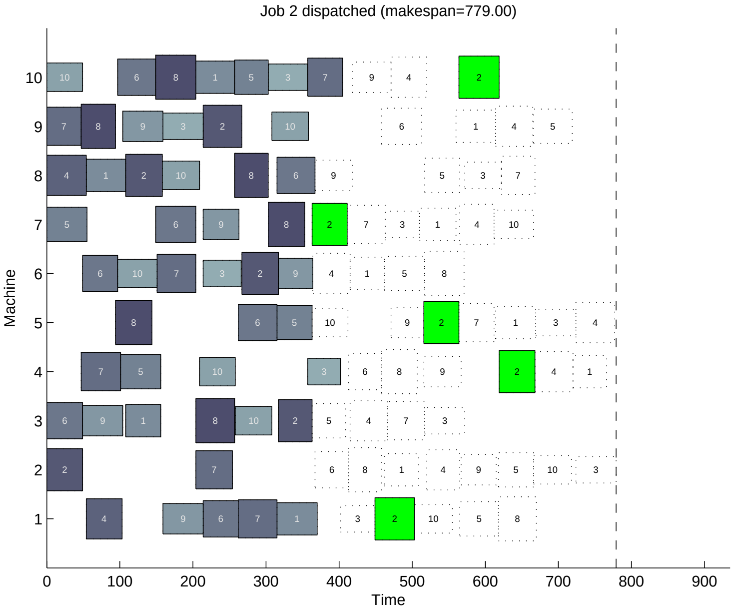
Jobshop-10 $\times$ 10, $U(1, 100)$ – impact on objective



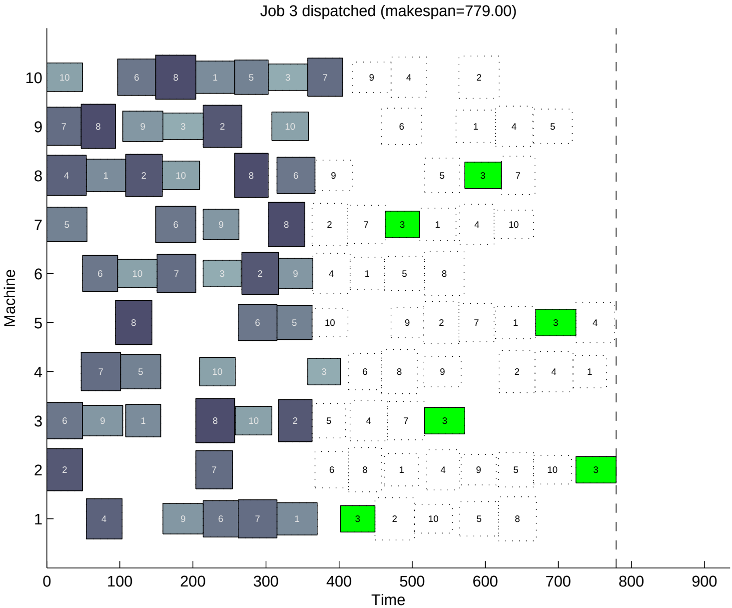
Jobshop processing times $U(51, 100)$ order random



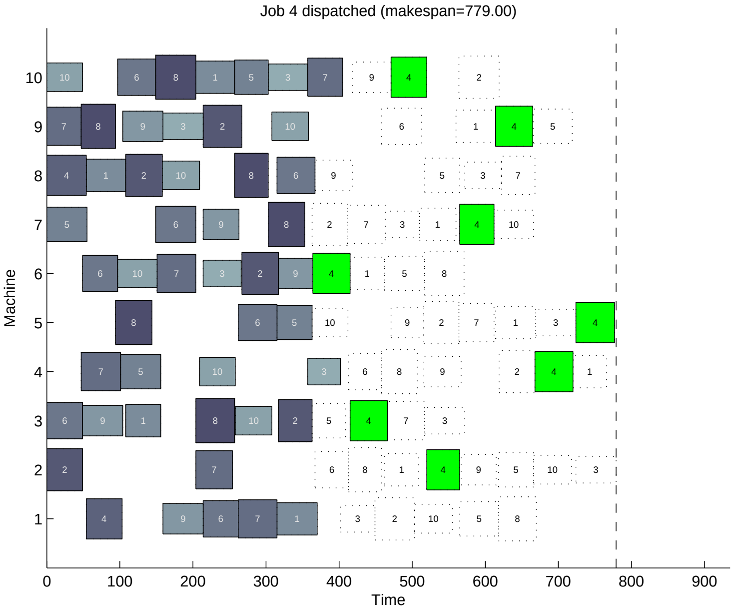
Jobshop processing times $U(51, 100)$ order random



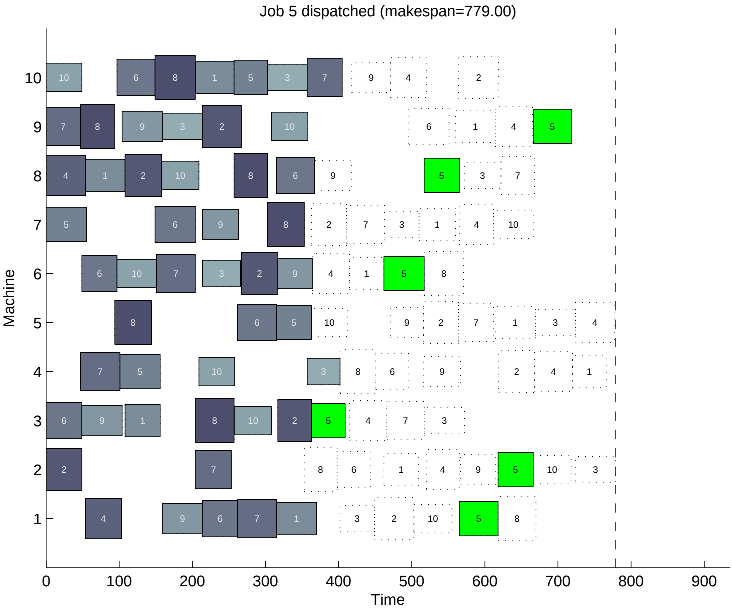
Jobshop processing times $U(51, 100)$ order random



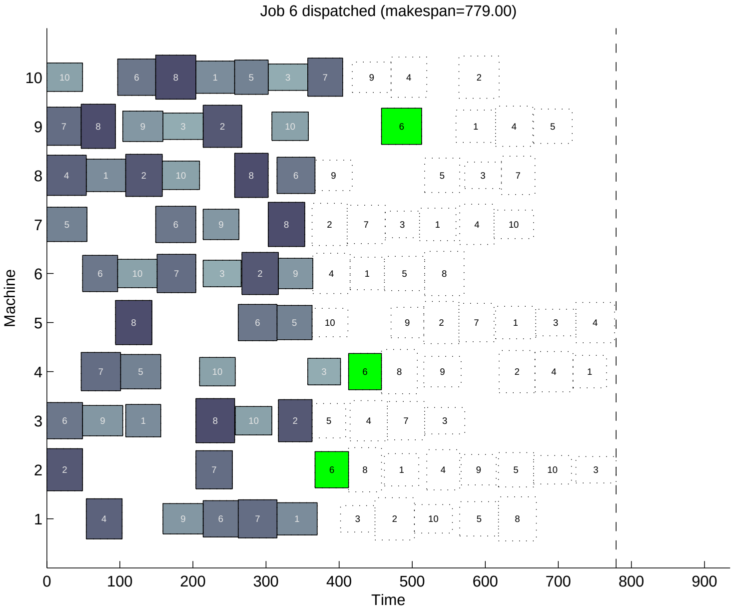
Jobshop processing times $U(51, 100)$ order random



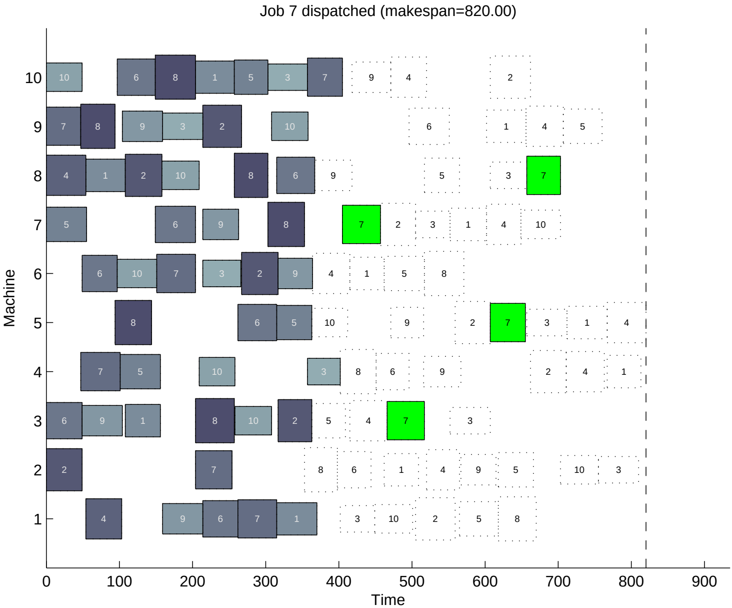
Jobshop processing times $U(51, 100)$ order random



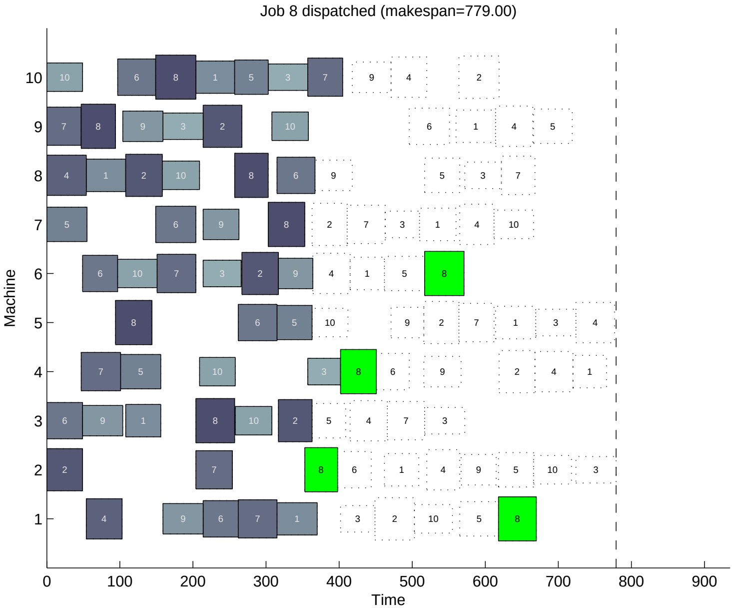
Jobshop processing times $U(51, 100)$ order random



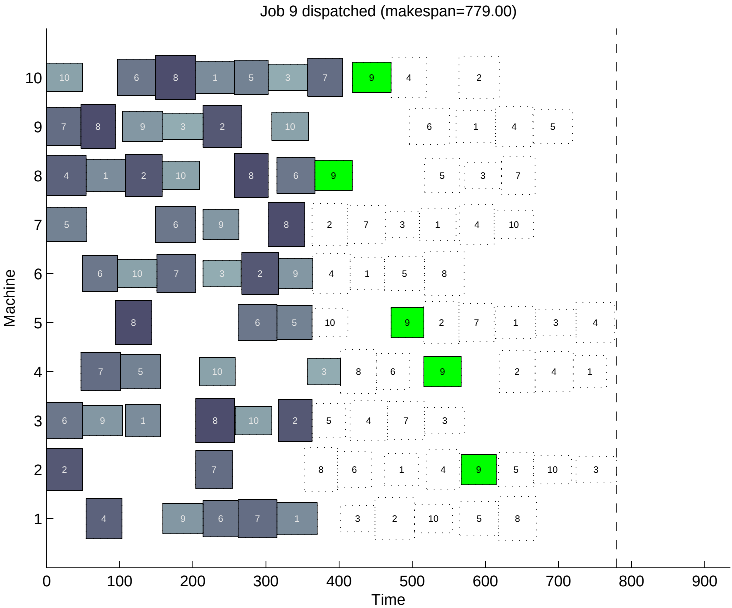
Jobshop processing times $U(51, 100)$ order random



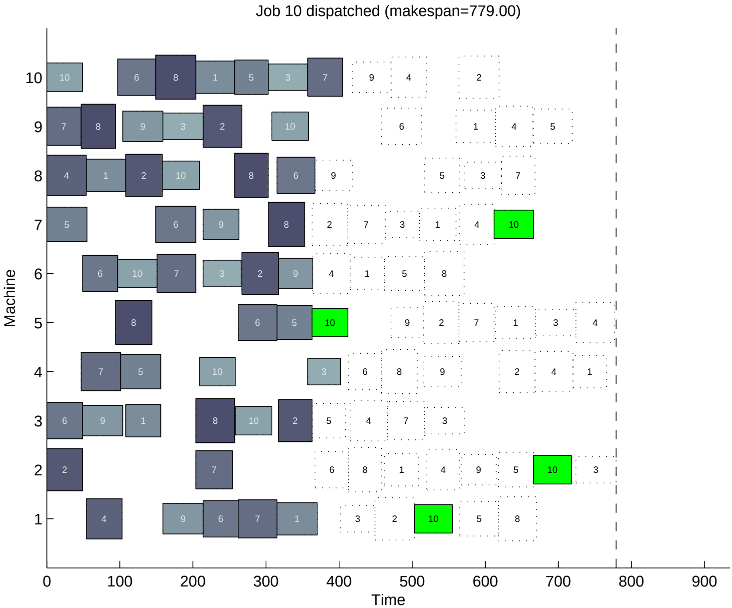
Jobshop processing times $U(51, 100)$ order random



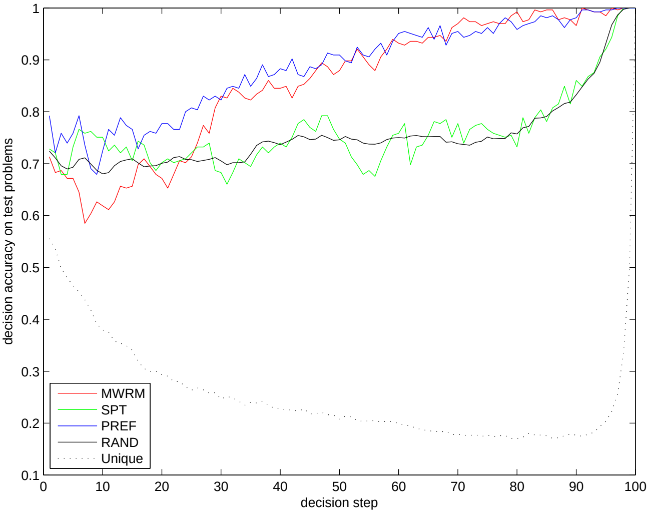
Jobshop processing times $U(51, 100)$ order random



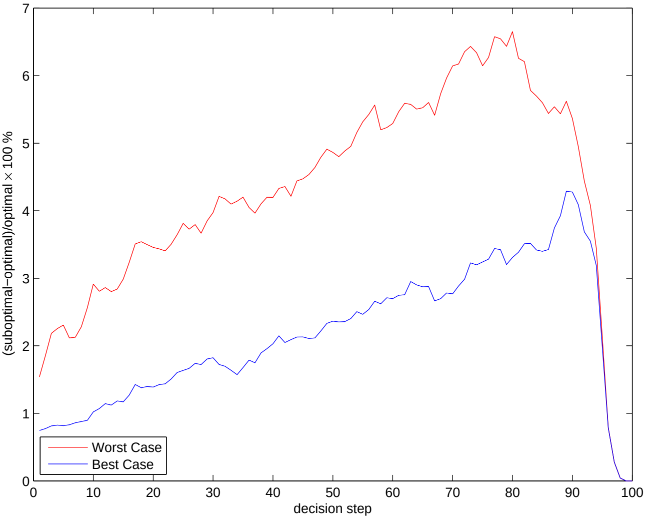
Jobshop processing times $U(51, 100)$ order random



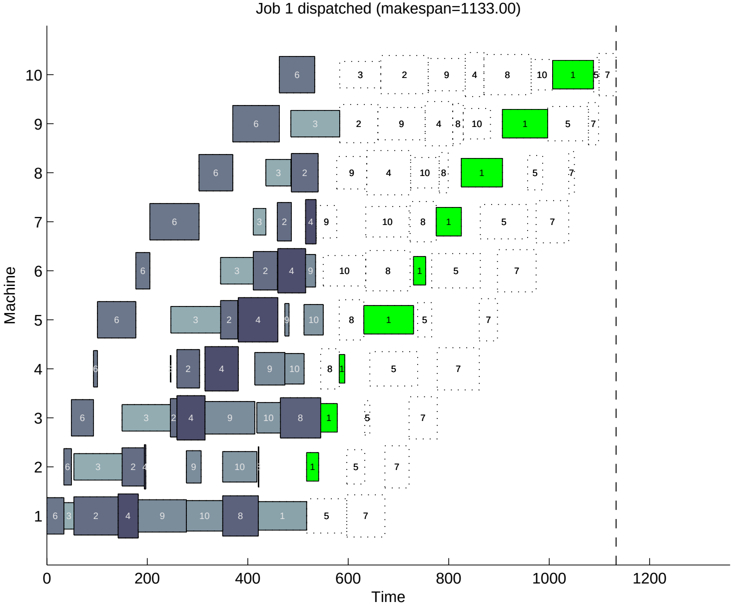
Jobshop-10 × 10, $U(51, 100)$ – average decision accuracy



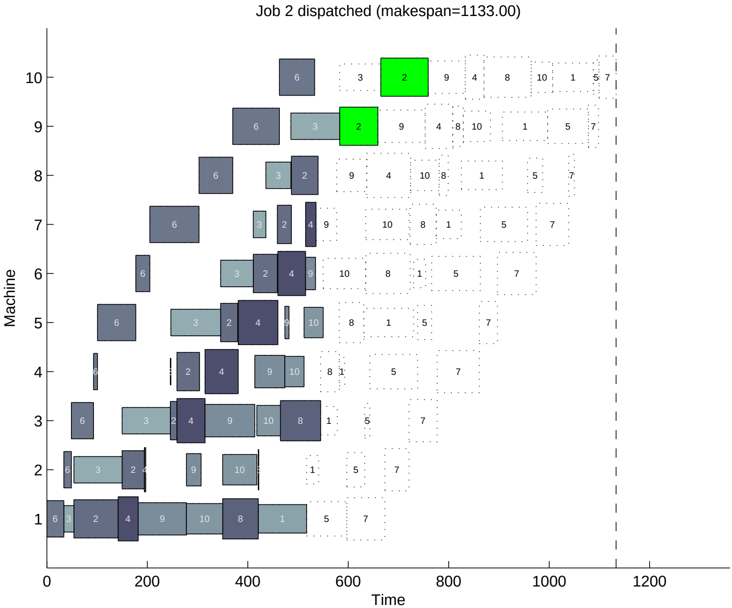
Jobshop-10×10, $U(51, 100)$ – impact on objective



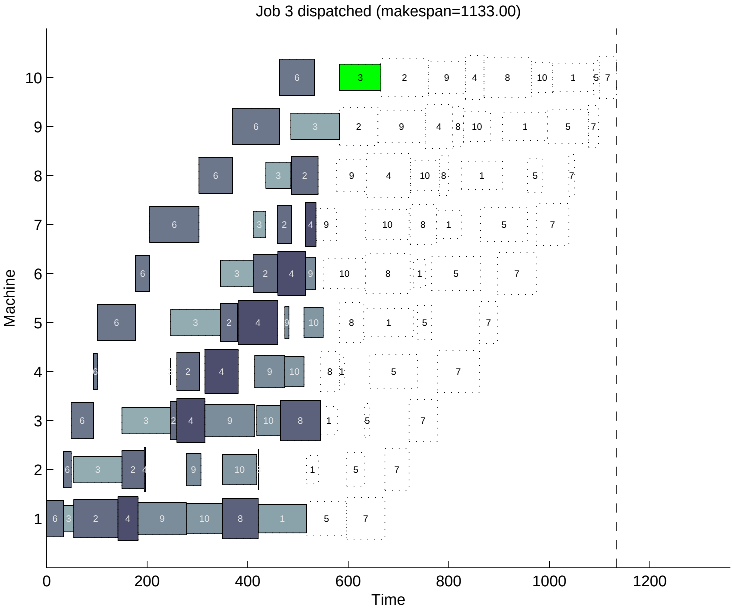
Permutation flow shop with processing times $U(1, 100)$



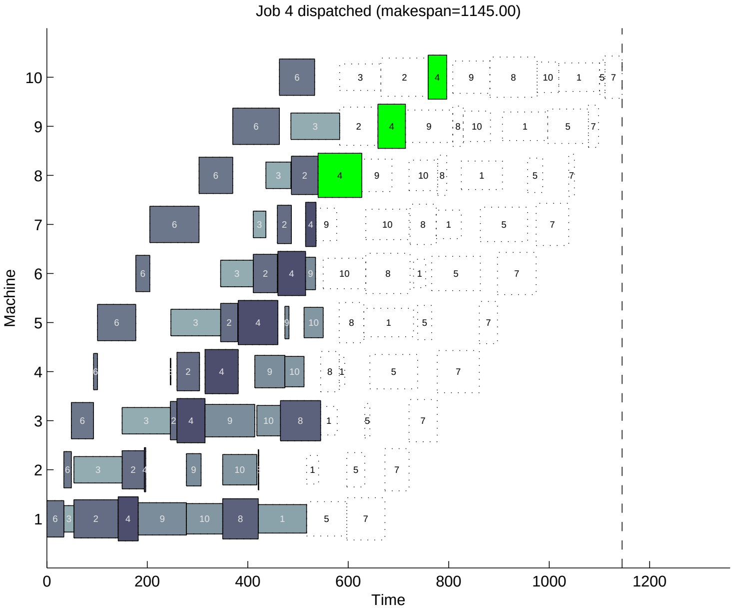
Permutation flow shop with processing times $U(1, 100)$



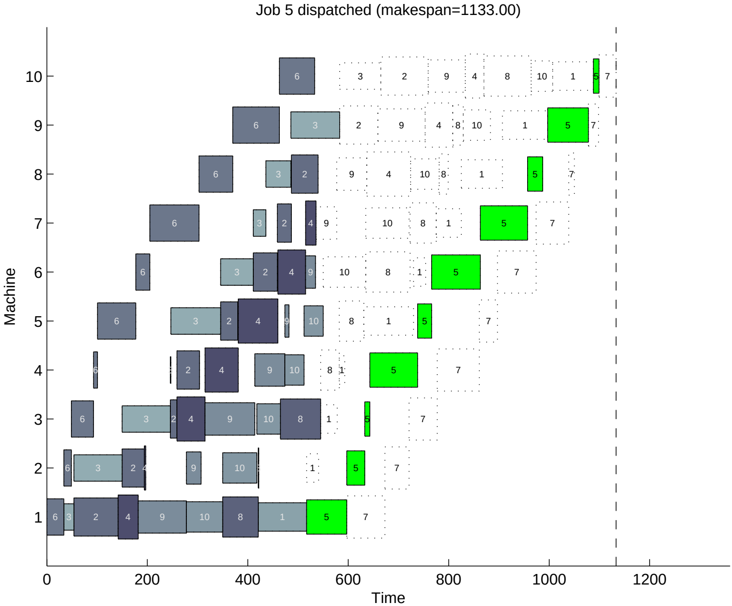
Permutation flow shop with processing times $U(1, 100)$



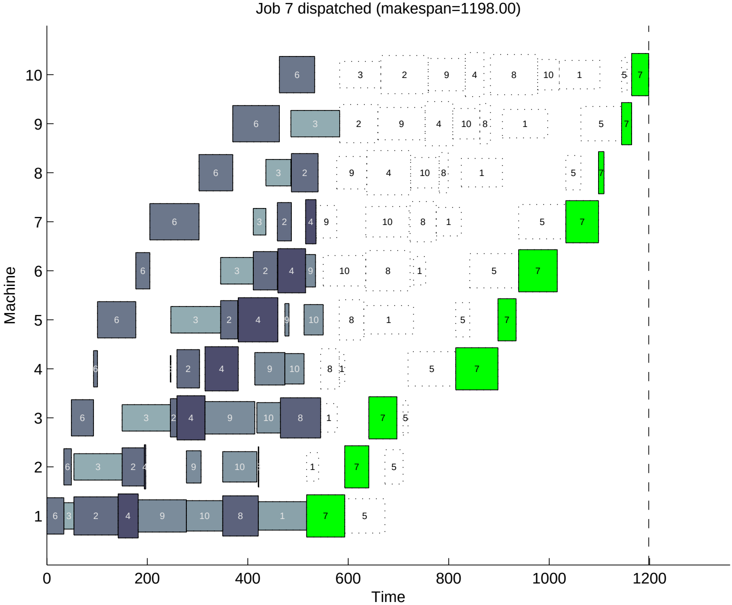
Permutation flow shop with processing times $U(1, 100)$



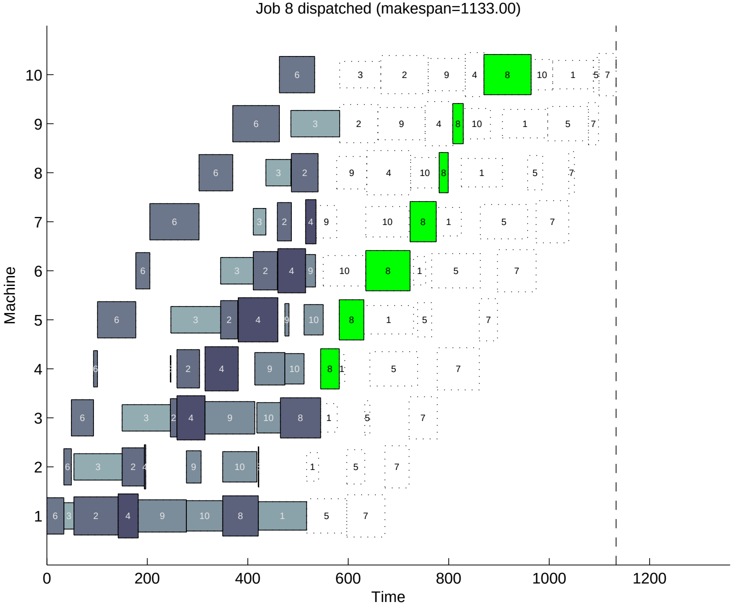
Permutation flow shop with processing times $U(1, 100)$



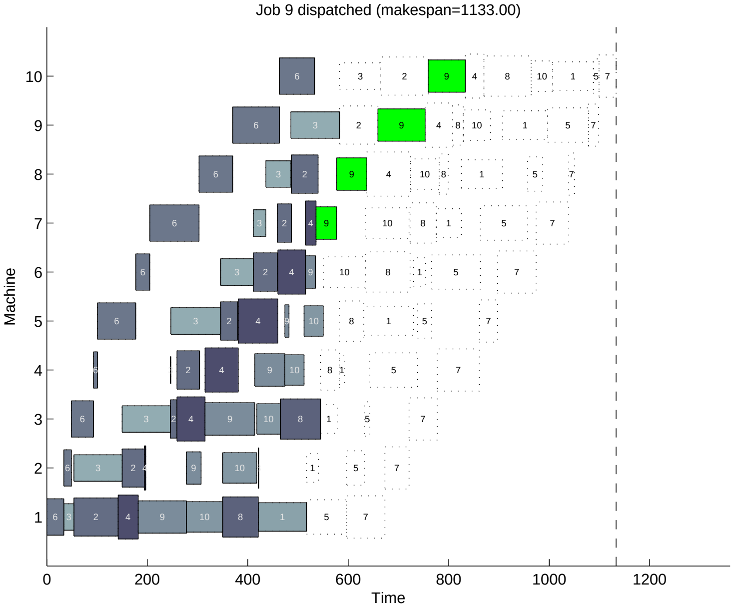
Permutation flow shop with processing times $U(1, 100)$



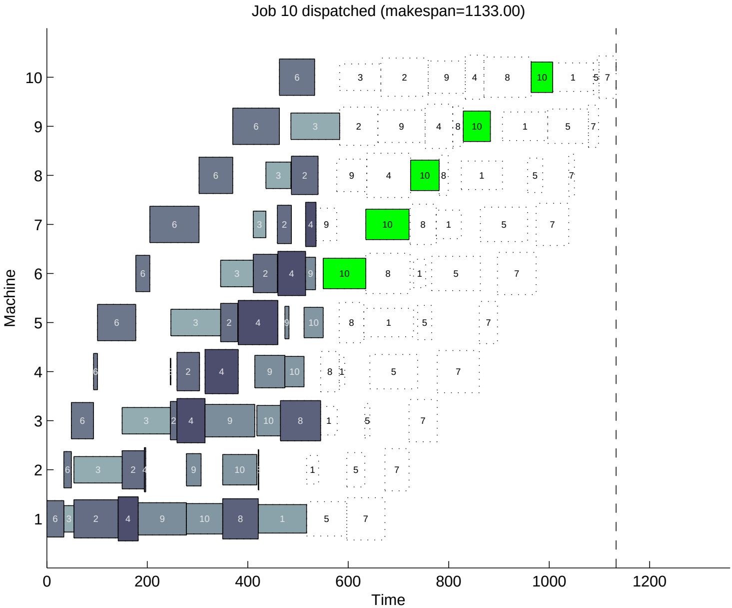
Permutation flow shop with processing times $U(1, 100)$



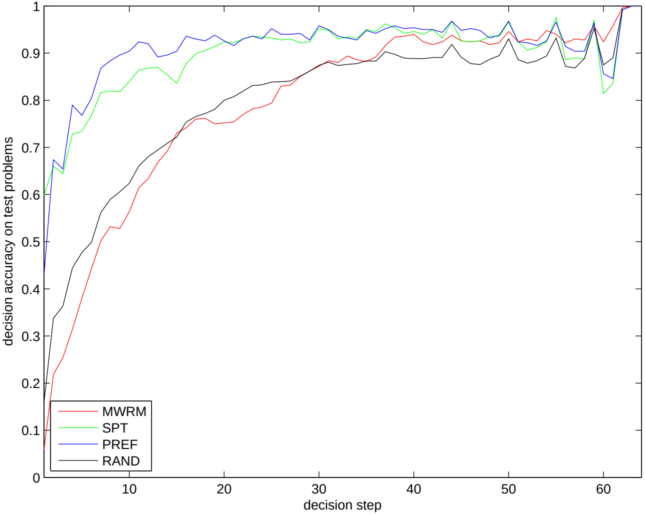
Permutation flow shop with processing times $U(1, 100)$



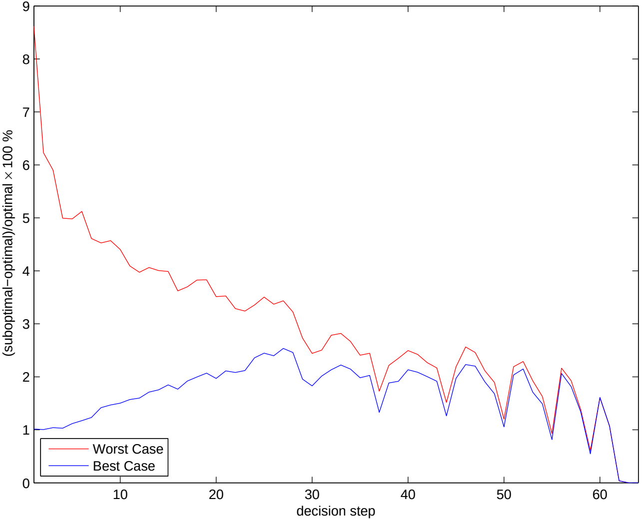
Permutation flow shop with processing times $U(1, 100)$



Flowshop-8 $\times$ 8, $U(1, 100)$ – average decision accuracy



Flowshop-8 $\times$ 8, $U(1, 100)$ – impact on objective



Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.

- Problem specific feature discovery for combinatorial optimization.

- Understanding where decision are critical for success (focus the training data).

- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.

- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.

Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.
- Problem specific feature discovery for combinatorial optimization.
- Understanding where decision are critical for success (focus the training data).
- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.
- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.

Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.
- Problem specific feature discovery for combinatorial optimization.
- Understanding where decision are critical for success (focus the training data).
- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.
- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.

Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.
- Problem specific feature discovery for combinatorial optimization.
- Understanding where decision are critical for success (focus the training data).
- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.
- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.

Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.
- Problem specific feature discovery for combinatorial optimization.
- Understanding where decision are critical for success (focus the training data).
- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.
- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.

Discussion and Challanges

- Visualization techniques (analytics) for combinatorial optimization.
- Problem specific feature discovery for combinatorial optimization.
- Understanding where decision are critical for success (focus the training data).
- Sub-optimal trajectory sampling? For games we have found that sampling more effectively the game state space is important.
- Apply these techniques directly within MIP solvers such as Gurobi. We are currently investigating this using SCIP.