Cooperative Games with Monte Carlo Tree Search

CheeChian Cheng and Norman Carver
Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA

Abstract—Monte Carlo Tree Search approach with Pareto optimality and pocket algorithm is used to solve and optimize the multi-objective constraint-based staff scheduling problem. The proposed approach has a two-stage selection strategy and the experimental results show that the approach is able to produce solutions for cooperative games.

Keywords: Monte Carlo Tree Search, Multi-Objective Optimization, Artificial Intelligence, Cooperative Game.

1. Introduction

Monte Carlo Tree Search (MCTS) is thriving on zero-sum games like the Go board game and there are many MCTS extensions that further optimize the selection strategy and improve the performance of the algorithm, e.g. RAVE [5]. In this paper, we are proposing to use MCTS to solve cooperative games and utilize the Staff Scheduling problem to show that the approach is able to produce solutions for cooperative games. The objective of a cooperative game is to maximize the team’s utility while at the same time optimizing the individuals utilities.

1.1 State-Of-The-Art

Optimum or near optimum solutions to the staff scheduling problem can be found by various approaches like Genetic Algorithm, Constraint Programming, Integer Programming and Single Player Monte Carlo Tree Search. As pointed out in the Single Player MCTS approach [4], the MCTS is easy to implement and the algorithm can be terminated if it runs out of resources, e.g. the search will end if it runs out of time.

2. Background Work

2.1 Tree

A tree structure is a data structure in computer science that represents information in a hierarchical manner (non-linear) resembling a tree as shown in Fig 1 [3]. Information is stored in tree nodes. A tree node can be a root node, a branch node or a leaf node. A root node is a node that does not have a parent node. A branch node is a node that has a parent node and has at least one child node. A leaf node is a node with a parent node but without a child node. Root, branch and leaf nodes are labeled as A, B and C respectively in Fig 1.

2.2 MiniMax Tree

A minimax tree is a tree structure (in decision theory and game theory fields) that represents a zero-sum game’s utilities as shown in Fig. 2 [7]. A zero-sum game is a competitive game where players are competing against each other to win the game, i.e. the gain of one player is the loss to other players. A minimax strategy is the strategy to minimize possible loss to a player when the player is making a decision.

2.3 Multi-armed Bandit

The objective of Multi-armed Bandit is to find the optimal strategy for a gambler to pull the levers of a row of slot machines in order to gain maximum reward. It is measured by the regret ρ as shown in Eq. 1.

\[ \rho = T\mu^* - \sum_{t=1}^{T} r_t \]  

where \( T \) is the number of rounds the gambler has pulled the levers, \( \mu^* \) is the maximal reward mean and \( r_t \) is the reward at time \( t \). The regret, \( \rho \) is the difference between the reward for optimal strategy and the total collected rewards after \( T \) rounds [1]. The average regret is \( \rho/T \) and it can be minimized if the gambler played enough rounds as \( \lim_{T \to \infty} \frac{\rho}{T} \to 0 \).

2.4 Monte Carlo Tree Search

Monte Carlo Tree Search has been used with various problems, including but not limited to, the Go board game,
real time strategy games, platform video games and staff scheduling optimization. Monte Carlo Tree Search consists of four (4) phases [2], they are:

1) Selection
2) Expansion
3) Simulation
4) Back propagation

In the Selection phase (Fig. 3), the best UCT (as shown in Eq. 3) of a tree node is selected by working recursively from the root node of the tree until a leaf node is reached. The leaf node (or terminal node) will be selected as the candidate for next phase. In the Expansion phase, a legal move will be randomly (uniform distribution) selected from all the possible moves based on the selected node (from selection phase). A new node (resulting from the move) will be added to the selected node as a child node as shown in Fig. 4. The simulation phase will begin with randomly sampling the possible moves for the new node (as shown in Fig. 5) until it reaches a terminal condition, e.g. until the game is over or all players can no longer make a move. The reward is then calculated and back propagated from the simulation phase’s terminal node up to the root node of the Monte Carlo tree as shown in Fig. 6. While traversing through each node during the back propagation phase, the visit counter in each node is incremented by one to indicate how many times it has been visited.

2.5 Pareto Optimal

In game theory, Pareto Optimality is a situation where one’s utility cannot be improved without degrading others’ utility. A Pareto front consists of all the Pareto optimal points, as shown in Fig. 7 [8].

3. The Problem

Creating a monthly schedule for staffing is a frequent task for schedulers in any human resource centric enterprise. Keeping the staff happy is extremely important in a human resource centric enterprise as it boosts the morale of the staff, improves productivity and the staff retention rate. Staff schedule preferences are soft constraints for scheduling, which the schedulers will try their best to accommodate. Hard constraints are the rules that cannot be broken, for instance, double booking a staff, back-to-back shifts and not honoring staff’s off day request.

3.1 Hard Constraints

Hard constraints are the rules that the schedulers cannot violate. If the scheduler breaks the hard constraints, the solution/schedule is considered not feasible. The hard constraints are:

- Hard Constraint #1 - Each staff must work at least the minimum contract hours. (HC#1)
- Hard Constraint #2 - It must be at least 12 hours apart between any two shifts for a staff to work in. (HC#2)
- Hard Constraint #3 - Scheduler cannot assign a shift to a staff on his/her requested off-day. (HC#3)

3.2 Soft Constraints

Soft constraints are the constraints that the schedulers will try to accommodate as much as possible. The soft constraints are:

- Soft Constraint #1 - Staff’s work day preference, i.e. weekday or weekend. (SC#1)
- Soft Constraint #2 - Staff’s shift preference. (SC#2)

4. Proposed Approach

We propose to use Monte Carlo Tree Search with Pareto optimality and pocket algorithm utilizing the multi agents approach. Each agent in turn makes its move by either exploring or exploiting its current situation. Each tree node
in the Monte Carlo Tree has a utility vector that consists of the team’s utility and each agent’s utility as shown in Eq. 2. Each phase of Monte Carlo Tree Search is described in the following subsections.

4.1 Utility Vector

Utility vector, as shown in Eq. 2 contains the utility for each agent and the team. The $u_{\text{team}}$ is the utility for the whole team and contains the value in the range of 0 and 1. The $u_{\text{team}}$ indicates whether the solution is a feasible solution or a non-feasible solution. The $u_{\text{team}}$ will be set to 1 if the solution is feasible otherwise it will be set to 0. A feasible solution is a solution where all agents’ assignments do not violate the hard constraints. $u_i$ is the utility for the agent $i$ and it is in the range of 0 and 1. The agent’s utility is the measurement of how well the solution is accommodating to the agent’s preferences, i.e. soft constraints.

$$\vec{U} = [u_{\text{team}}, u_1, u_2, ..., u_n]$$  \hspace{1cm} (2)

where $u_{\text{team}}$ is the utility for the team, $u_i$ is the utility for agent $i$.

4.2 Selection

During the selection phase, each agent in turn make its move. When it is the agent’s turn to make a move, the agent will select a tree node in 2 sub phases.

1) team utility selection
2) agent’s utility selection

Starting from the root of the tree, a tree node will recursively be selected, until a tree leaf is reached. In the first phase, the tree node with the highest mean team utility will be selected if it is not fully explored. If there are multiple tree nodes, the node with the highest mean utility for the agent will be selected. The tree node is being selected for
the next phase (simulation). If there is no possible moves for the agent from the selected node, the next agent will be chosen to make a move. A node will be labeled as a terminal node if no agents can make a move from it. The UCT for a team or an agent $i$ is shown in Eq. 3 [6].

$$\text{UCT} = \sum X_j \frac{1}{n_j} + 2C \sqrt{\frac{\ln(n)}{n_j}}$$

where $\sum X_j$ is the total utility for the team or agent for the child node $j$, $n_j$ is the number of visits count in the child node $j$, $n$ is the number of visits count for the parent node, and $C$ is a constant.

Each tree node has the following properties:
- Agent
- Move
- Number of visits
- Utility vector

The node’s “Agent” indicates which agent’s turn it is to make a move, while “Move” is the move that the agent has made. “Number of visits” is a counter indicating the number of times the tree node has been visited during the simulation phase. The “Utility vector” keeps track of the team and individual agent utilities.

Algorithm 1 Selection

<table>
<thead>
<tr>
<th>INPUT: TreeNode, AgentList, CurrentAgent</th>
<th>OUTPUT: SelectedNode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurretNode = TreeNode</td>
<td></td>
</tr>
<tr>
<td>while CurrentNode is not leaf do</td>
<td></td>
</tr>
<tr>
<td>Candidates = tree nodes with highest $\text{UCT}_{\text{team}}$ that are not fully explored</td>
<td></td>
</tr>
<tr>
<td>if there are multiple candidates then</td>
<td></td>
</tr>
<tr>
<td>Candidates = Candidates with the highest $\text{UCT}_j$</td>
<td></td>
</tr>
<tr>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>if no Candidates then</td>
<td></td>
</tr>
<tr>
<td>Indicate the tree is fully explored and stop the search</td>
<td></td>
</tr>
<tr>
<td>return null</td>
<td></td>
</tr>
<tr>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>CurrentNode = randomly (uniform) pick one of the Candidates</td>
<td></td>
</tr>
<tr>
<td>end while</td>
<td></td>
</tr>
<tr>
<td>return CurrentNode</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Simulation

In order to harness the advantage of the multi-armed bandit model [1], the simulation phase will simulate all possible moves available to the agent. An agent will be skipped if the agent can no longer make a move from the node. The simulation phase will cease when no agents can make any moves. At the end of each simulation phase, the utilities for the team and the individual agents are computed.

Algorithm 2 Simulation

<table>
<thead>
<tr>
<th>INPUT: TreeNode, AgentList</th>
<th>OUTPUT: SubTrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialAgent = TreeNode.Agent</td>
<td></td>
</tr>
<tr>
<td>result = [] // empty</td>
<td></td>
</tr>
<tr>
<td>Get all possible moves for this tree node</td>
<td></td>
</tr>
<tr>
<td>for each move in possible moves do</td>
<td></td>
</tr>
<tr>
<td>NextAgent = next agent (round-robin based on current agent and agent list)</td>
<td></td>
</tr>
<tr>
<td>NewTreeNode = new tree node with agent and move</td>
<td></td>
</tr>
<tr>
<td>SubTree = Sample(NewTreeNode, AgentList)</td>
<td></td>
</tr>
<tr>
<td>Append SubTree to result (as an array)</td>
<td></td>
</tr>
<tr>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>return result</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm 3 Sample

<table>
<thead>
<tr>
<th>INPUT: TreeNode, AgentList</th>
<th>OUTPUT: SubTree</th>
</tr>
</thead>
<tbody>
<tr>
<td>NextAgent = next agent of TreeNode.Agent (round-robin)</td>
<td></td>
</tr>
<tr>
<td>Get all possible moves for this tree node and current agent</td>
<td></td>
</tr>
<tr>
<td>CurrentAgent = NextAgent</td>
<td></td>
</tr>
<tr>
<td>NewMove = move</td>
<td></td>
</tr>
<tr>
<td>repeat</td>
<td></td>
</tr>
<tr>
<td>Create a tree node to represent the NewMove and CurrentAgent</td>
<td></td>
</tr>
<tr>
<td>Add the NewNode as a child to CurrentNode</td>
<td></td>
</tr>
<tr>
<td>CurrentNode = NewNode</td>
<td></td>
</tr>
<tr>
<td>CurrentAgent = next agent of CurrentAgent (round-robin)</td>
<td></td>
</tr>
<tr>
<td>Get all possible moves for CurrentNode and CurrentAgent</td>
<td></td>
</tr>
<tr>
<td>NewMove = randomly (uniform) pick one of the possible move</td>
<td></td>
</tr>
<tr>
<td>until no agents can make a move</td>
<td></td>
</tr>
<tr>
<td>Calculate utilities for current node</td>
<td></td>
</tr>
<tr>
<td>return TreeNode</td>
<td></td>
</tr>
</tbody>
</table>

4.4 Expansion

The results from the simulations (as sub-trees) will be merged into the Monte Carlo Tree. Each simulation result will be tested for Pareto optimality. The Pocket algorithm is used to remember the Pareto optimal solutions.
Algorithm 4 Pocket Algorithm

INPUT: Pocket, terminal nodes

for each node in terminal nodes do
    Remove the solutions from the Pocket that are suboptimal to node’s utility vector
    if node’s utility vector is not sub-optimal among the solutions from the Pocket then
        Add node to the Pocket
    end if
end for

4.5 Pocket Algorithm

Pocket algorithm consists of two (2) phases, they are:
1) Removing suboptimal solutions
2) Adding solutions from the simulation phase to the solution pool (also known as pocket)

In phase 1, each node from the simulations phase is compared to all the nodes in the solution pool. The nodes in the solution pool will be removed if they are suboptimal to the nodes from the simulation phase. In phase 2, each node from the simulation phase is compared to all the nodes in the solution pool and the node from the simulation phase will be added to the solution pool if they are not suboptimal to any solutions in the pocket.

4.6 Back Propagation

The utility vector is propagated from the terminal node back up to the root node of the Monte Carlo Tree. The tree node will be marked as fully explored if all child nodes have been explored. The leaf node (or terminal node) will always be marked as fully explored.

Algorithm 5 Back Propagation

INPUT: node

$\vec{U}$ to node’s utility vector
Node = parent node
while node is not root do
    Add $\vec{U}$ to node’s utility vector
    Increment node’s visit counter
    Node = parent node
end while

5. Experiments and Results

The proposed approach has been applied to the constraints based staff scheduling problem, where each staff viewed as an agent and the shifts being the moves that the agents can make. Each agent has its own constraints including off days, preferred working shifts and preferred working days. Table 1 shows the preferences and their respective off-day requests for 3 staff members.

<table>
<thead>
<tr>
<th>Staff</th>
<th>Off Day</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>1st day of the month</td>
<td>Prefer to work during week days</td>
</tr>
<tr>
<td>#2</td>
<td>2nd day of the month</td>
<td>Prefer to work during weekend</td>
</tr>
</tbody>
</table>

Several experiments were conducted with the criteria in Table 1. The results are discussed in the following subsection. All experiments were run with Internet Explorer 11 on an Intel Xeon 3.4GHz, 8GB RAM. The programs were written in typescript (a superset of javascript). The utility for each agent is computed as per Eq. 4. Each experiment was run with 100 simulations and $C$ was set to 1.414.

Table 1: Experiment Criteria.

Table 2: Experiments.

<table>
<thead>
<tr>
<th>Days</th>
<th>Item</th>
<th>Day</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Thursday</td>
<td>Jan/1/15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Friday</td>
<td>Jan/2/15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Friday</td>
<td>Jan/2/15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Saturday</td>
<td>Jan/3/15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thursday</td>
<td>Jan/4/15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Friday</td>
<td>Jan/2/15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Saturday</td>
<td>Jan/3/15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sunday</td>
<td>Jan/4/15</td>
<td></td>
</tr>
</tbody>
</table>

$u_i = 0.5 \times N$  \hspace{1cm} (4)

where N is the number of soft rules that comply with the preferences of agent i.

5.1 Result Discussion

A feasible solution is the solution that does not violate any hard rules. Fig. 9 and Fig. 11 show the feasible solutions for the 3-day and 4-day schedules respectively. Fig. 10 and Fig. 12 show the solutions that violated the hard rules and so are considered not feasible. For instance, in the 4-day
Reward Vector = [1, 0.50, 0.25, 0.33]
Euclidean distance = 0.6508541396588878
Player = 2 Thu 1/1 7:0 - 1/1 19:0
Player = 2 Sat 1/3 7:0 - 1/3 19:0
Player = 1 Fri 1/2 7:0 - 1/2 19:0
Player = 3 Thu 1/1 19:0 - 1/2 7:0
Player = 3 Sat 1/3 19:0 - 1/4 7:0
Player = 3 Fri 1/2 19:0 - 1/3 7:0
Fig. 9: Solution to 3-day Schedule with $U_{team} = 1$.

Reward Vector = [0, 0.50, 0.50, 0.50]
Euclidean distance = 0.8600254037844386
Player = 1 Thu 1/1 19:0 - 1/2 7:0
Player = 2 Sat 1/3 19:0 - 1/4 7:0
Player = 1 Fri 1/2 7:0 - 1/2 19:0
Player = 3 Fri 1/2 19:0 - 1/3 7:0
Player = 2 Sat 1/3 7:0 - 1/3 19:0
Player = 1 Thu 1/1 7:0 - 1/1 19:0
Fig. 10: Solution to 3-day Schedule with $U_{team} = 0$.

Reward Vector = [1, 0.50, 0.33, 0.25]
Euclidean distance = 0.6508541396588878
Player = 3 Thu 1/1 19:0 - 1/2 7:0
Player = 3 Fri 1/2 19:0 - 1/3 7:0
Player = 3 Sat 1/3 19:0 - 1/4 7:0
Player = 2 Sun 1/4 7:0 - 1/4 19:0
Player = 2 Thu 1/1 7:0 - 1/1 19:0
Player = 3 Sun 1/4 19:0 - 1/5 7:0
Player = 2 Sat 1/3 7:0 - 1/3 19:0
Player = 1 Fri 1/2 7:0 - 1/2 19:0
Fig. 11: Solution to 4-day Schedule with $U_{team} = 1$.

Reward Vector = [0, 0.50, 0.50, 0.38]
Euclidean distance = 0.8003905296791061
Player = 3 Fri 1/2 19:0 - 1/3 7:0
Player = 2 Sun 1/4 19:0 - 1/5 7:0
Player = 3 Sun 1/4 7:0 - 1/4 19:0
Player = 3 Fri 1/2 7:0 - 1/2 19:0
Player = 1 Thu 1/1 7:0 - 1/1 19:0
Player = 3 Thu 1/1 19:0 - 1/2 7:0
Player = 2 Sat 1/3 7:0 - 1/3 19:0
Player = 2 Sat 1/3 19:0 - 1/4 7:0
Fig. 12: Solution to 4-day Schedule with $U_{team} = 0$.

schedule solution in Fig. 12, Staff #1 has an off day request for Jan/1 but the schedule indicates that Staff #1 has to work on Jan/1 from 7am-7pm, thus violating hard constraint HC#3. Take note, however, that the solution does align with the preferences of each staff, i.e. the proposed approach tried to schedule individual staff such that the schedule tends to accommodate with the preferences of individual staff. In Fig. 8, Staff #2 has a zero utility as the solution does not accommodate any of his/her preferences.

6. Conclusion

MCTS has been thriving on competitive zero-sum games like Go. In this paper, we have proposed an approach to solve and optimize cooperative games with MCTS using Pareto optimality and pocket algorithm. Unlike the approach from [4] that used a scalar value (scoring function) for each solution, the solutions from this new approach always align to the soft constraints, regardless of the hard constraints.

References