Constraint Based Staff Scheduling Optimization
Using Single Player Monte Carlo Tree Search

CheeChian Cheng, Norman Carver and Shahram Rahimi
Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA
cccheng@siu.edu; carver@cs.siu.edu; rahimi@cs.siu.edu

Keywords: Monte Carlo Tree Search, Artificial Intelligence, Optimization, Roster, Staff Scheduling.

Abstract: Single Player Monte Carlo Tree Search approach is used to solve and optimize the constraint-based staff scheduling problem for emergency department in health care industry and the results are compared with the optimum results obtained by brute force search.

1 INTRODUCTION

Optimization is a process of finding the highest achievable performance under a given set of constraints. There are various applications in optimization and one of the applications is to optimize the staff scheduling problem. The schedulers[1] for Critical Care Medicine unit (ER) are constantly facing the challenge of producing rosters that are filled up with providers[2] and at the same time, trying to fulfill every provider's preferences. However, the schedulers' main priority is to ensure that all shifts are filled up with providers. Thus, putting providers' preferences in lower priority, most of the time the rosters hardly meet the providers' preferences. In order to meet the providers' preferences, the schedulers have to spend an average of 25 man-hours to create and optimize the roster manually as staff scheduling is an NP-hard problem (Sotskov and Shakhlevich, 1995).

In this paper, *Monte Carlo Tree Search* (MCTS) is evaluated to solve and optimize the staff scheduling problem in the Health care industry. Several experiments are described and the results of the proposed MCTS approach are compared with the results of *brute force search* (BFS).

1.1 State-Of-The-Art

Throughout the years, various approaches have been proposed to optimize staff scheduling problem including *genetic algorithm* (GA), constraint programming techniques, greedy algorithms and other hybrid models. In recent work, Burke et al. has proposed a hybrid model of *integer programing* (IP) and *variable neighborhood search* (VNS) to optimize nurse rostering with hard and soft constraints (Burke et al., 2010). The article compared the hybrid VNS model with hybrid genetic algorithm model and they claimed that the hybrid VNS model outperformed the hybrid GA model.

Another hybrid model of IP and *Evolutionary Algorithm* (EA) was proposed and the authors claimed that the hybrid model of IP and EA outperformed the earlier model of IP and VNS (Huang et al., 2014).

Genetic algorithm is a popular approach to the scheduling problem and one of the recent work was done by Yang and Wu (Yang and Wu, 2012). They have reported that with the appropriate cross over and mutation operators, the GA method could find optimal or near-optimal work schedules.

Brucker et al. has proposed 2-stage local greedy search to assign as many shifts to the most constrained nurses as possible (Brucker et al., 2010). This may produce a feasible roster but may not produce an (near) optimum roster since the roster might be a local minimum instead of global minimum.

The proposed MCTS approach will shorten the search time significantly while producing an optimum (or near optimum) schedule by taking in all the constraints and preferences. The proposed approach is relatively easy to implement and it does not need any domain knowledge in staff scheduling. However, it does need to know the constraints or rules of the problem so that MCTS will be able to make valid "moves". Moreover, MCTS can be interrupted any time and produces the best solution found so far.

[1]The scheduler is a person who is responsible for creating monthly roster for the providers.

[2]The provider is a person who works at the hospital/facility.

2 BACKGROUND WORK

MCTS is an incremental tree building algorithm using Monte Carlo methods (Browne et al., 2012). A typical 2-player (player A and player B) tree is shown in Fig. 1. The selection of the best (or most promising) node is based on the *upper confidence bound for tree* (UCT) (Kocsis and Szepesvári, 2006) derived from multiarmed bandit algorithm's upper confidence bounds (UCB) (Auer et al., 2002). The UCT as shown in equation 1 acts as a balancer to either explore new node or exploit current best node.

$$UCT = \overline{X} + C\sqrt{\frac{\ln N}{N_i}} \tag{1}$$

In MCTS, the *Selection Phase* is a process of choosing the best (or most promising) node using UCT equantion. It follows by the *Expansion Phase* where the creation of new node is determined by the tree policy and often, the tree policy is a uniform distribution of all possible "moves" (Monte Carlo methods). A new node is randomly chosen from the possible "moves" and later, added to the tree as the descendant of the most promising node mentioned earlier. Play out simulation (*Simulation Phase*) will begin after the *Expansion Phase* by sampling the outcome of the simulation. Often, the outcome of the simulation of a game uses the minimax algorithm with alternate moves and the outcome of the simulation is propagated back to the root node, also known as *Back Propagation Phase*. A 2-player minimax tree (as shown in Fig. 2) shows that player A wins the game if s/he adopts strategy X and loses the game if s/he adopts strategy Y.

Go has a huge search space and by utilizing MCTS, the computer player achieved impressive winning rate against professional players. However, these are 2-player games and there was a need to come up with a MCTS for single player games like puzzle games. *Single Player MCTS* (SP-MCTS) has a UCT as shown in equation 2 (Schadd et al., 2008). It has an additional term to act as a balancer to either explore or exploit current tree.

$$UCT = \overline{X} + C\sqrt{\frac{\ln N}{N_i}} + \sqrt{\frac{\sum X^2 + N_i\overline{X}^2 + D}{N_i}} \tag{2}$$

SP-MCTS was applied to the "re-entrance" scheduling problem and produced impressive results in optimizing the schedule (Matsumoto et al., 2010). However, the "re-entrance" scheduling problem does not have hard and soft constraints like the staff scheduling problem.

3 THE PROBLEM

In Critical Care Medicine unit, creating monthly rosters for providers (doctors and nurses) to work in various shifts are frequent tasks for a scheduler. Most of the time, schedulers have difficulty assigning providers to empty shifts while working out a roster that fulfills every providers' preferences. Each provider may send their working preferences and off-days to the scheduler on monthly basis before the scheduler starts creating the roster. In normal practice, the providers may request five (5) to seven (7) off days per month where they will not be scheduled to work on the off-days. Then, the scheduler would produce a feasible solution before trying to meet all the providers preferences. This approach may produce globally optimized solution (measurement of optimality is described in sub-section 4.6) but often produces locally optimized solution. Hence, there are two types of constraints, hard and soft constraints[3].

[3]Acknowledgment: This is to acknowledge that this work was supported by R&B SOFT LLC, providing information on rules and practices in the emergency departments and the required data for evaluation and testing purposes.

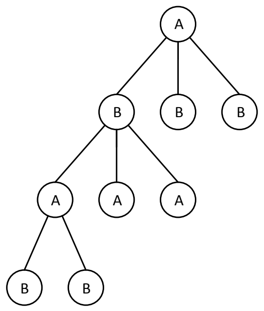


Figure 1: A 2-Player Tree.

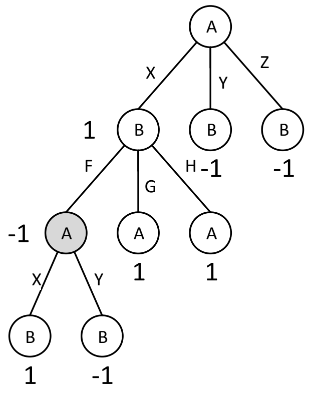


Figure 2: A 2-Player MiniMax Tree.

3.1 Hard Constraints

Hard constraints are the constraints that if the roster violates them, it is considered not a feasible solution/roster. The hard constraints listed below were used in the experiments.

- Hard Constraint #1 - Each provider must work at least the minimum contract hour. (HC#1)
- Hard Constraint #2 - There must be at least 12 hours apart between any two working shifts for the same provider. (HC#2)
- Hard Constraint #3 - A Provider's working shifts cannot collide with his/her off-days selection. (HC#3)

3.2 Soft Constraints

Soft constraints are the constraints that the roster should try to accommodate. The more the roster matches the soft constraints, the better (measured by the scoring function as described in sub-section 4.6) the roster is. The soft constraints listed below were used in the experiments.

- Soft Constraint #1 - Provider's preference to work on weekday or weekend. (SC#1)
- Soft Constraint #2 - Provider's working shift preference. (SC#2)
- Soft Constraint #3 - Weekend/weekday balancing for each provider. (SC#3)

4 PROPOSED APPROACH

The staff scheduling problem can be treated as a single player game and we are proposing the SP-MCTS (Schadd et al., 2008) with the UCT as shown in equation 2 to control the exploration and exploitation of the nodes in the tree. From equation 2, N_i is the number of times node i has been visited, N is the number of times the parent of node i has been visited, $\overline{X}$ is average score (refer to sub-section 4.6) of the node i ($\overline{X} = \frac{\sum X}{N_i}$), $\sum X^2$ (or $\|X\|^2$) is the sum of square score for the node i, C and D are constants. When building the tree, a node can be a branch node, terminal node or a fully explored branch node. Types of nodes are defined as follow:

1. Leaf node - the node has no children.
2. Branch node - the node has the potential to expand and may generate feasible solutions regardless of whether the node has children or not.
3. Terminal node - no more valid move can be made at this node and the node is a leaf node as well.
4. Fully explored branch node - this node has been fully explored as all valid moves have been evaluated.

Each node is regarded as a move and a move consists of the provider who is going to work in the shift (date and time). Generally, MCTS has four (4) phases (Browne et al., 2012):

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

However, in our proposed approach, we made a slight modification by adding another Expansion phase after Simulation phase, we name it Expansion2 phase.

4.1 Selection

During the selection phase, starting from the root node of the tree, each set of nodes (parent-children) will be evaluated and the node with the highest UCT from equation 2 will be selected (as shown in Fig. 3). In order to avoid over sampling, only branch nodes are evaluated. If the child node is selected, it will be set as the next parent and the next set of nodes will be evaluated again. This process continues until either the parent or the leaf node is selected. A higher value of C is preferred to encourage exploration instead of exploitation.

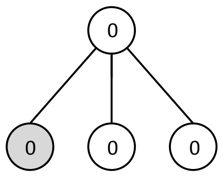


Figure 3: Selection with the nodes of score zero.

4.2 Expansion

A valid move is a move that does not violate hard constraints #2 and #3. During expansion phase, a valid move is randomly picked among the valid moves as next move as shown in Fig. 4. Another valid move will be randomly picked again if the selected move exists in the tree and the node is not a branch node. If there is no valid move, current node will be marked as fully explored branch if it has at least one child node.

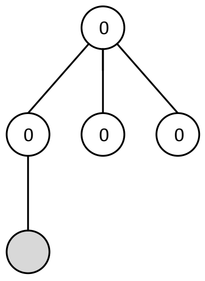
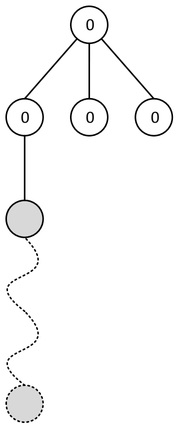


Figure 4: Expansion.

Otherwise, it will be marked as a terminal node. Next selection phase will be executed if it is marked as fully explored branch or terminal node.

4.3 Simulation

With the selected valid move from Expansion phase, a new roster is constructed from selected valid move up to the root node of the tree. Based on this newly constructed roster, the simulation phase will run a series of play out until there is no valid move can be made (Fig. 5).



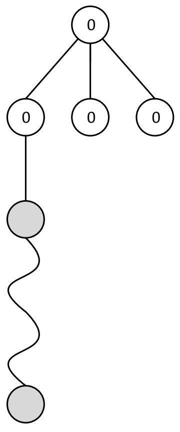


Figure 5: Simulation. Figure 6: Expansion2.

4.4 Expansion 2

All these play out moves from Simulation phase are appended to the tree as nodes as shown in Fig. 6. The reason behind this phase is to produce a feasible solution if user chooses to interrupt the algorithm. Without this phase, the algorithm will not be able to construct a feasible roster from the terminal node which has the highest score.

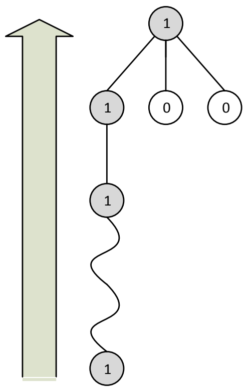


Figure 7: Back Propagation.

4.5 Back Propagation

The score of the simulation is calculated and back propagated from the terminal node up to the root node as shown in Fig. 7. The number of visitation will also be incremented as well. From equation 2, $\overline{X}$ is the average score for the node, $\sum X^2$ is the total sum square of the score, N_i is the number of visit of the node and N is the number of visit of the parent node.

4.6 Scoring Function

Scoring function is a vector of scores or rewards and the purpose of the scoring function is to measure the optimality of the schedule. This vector consists of a base score and other soft constraints scores as well. The base score will have a score of zero (0) if any of the provider does not meet the minimum contract hours (HC#1). Otherwise, a score of one (1) will be assigned to the scoring function. HC#2 and HC#3 have been addressed during Selection phase earlier, hence the scoring function does not have to deal with these constraints. Each soft constraint will have a score ranging from 0 to 0.1 depending on how well the solution fits the providers' preferences. An additional term, the soft constraints balancer score is the measurement of how well the soft constraint scores are uniformly distributed. This score vector will then be transformed into a single objective score as shown in equation 3.

$$Score = B \times (\overrightarrow{SC} \cdot \overrightarrow{W}^{\mathrm{T}}) \tag{3}$$

where B is the base score (hard constraints), $\overrightarrow{SC}$ is the score vector of soft constraints and $\overrightarrow{W}$ is the weight vector representing the importance of each soft constraint.

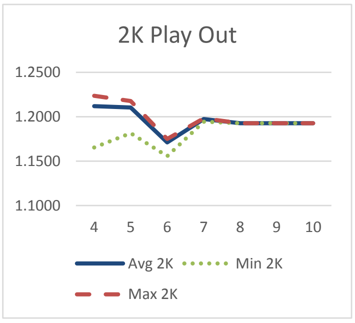


Figure 8: 2K Play Out Results.

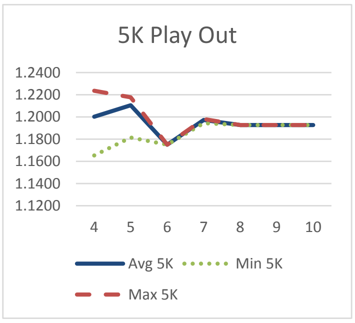


Figure 9: 5K Play Out Results.

5 EXPERIMENTS AND RESULTS

In our experiments, rosters of 1 shift per day with 3 providers and various numbers of days (4 to 10) were used as test cases. Table 1 shows the off day and preference for each provider.

Brute force search was used to find the highest score (optimum solution) and we compared the result of the proposed approach with the optimum solution from brute force search. All experiments were run on Intel i7 4770 3.4GHz with 12GB RAM and the programs were written in C#.

In Table 2, column #4 (the # of solutions) shows the number (count) of optimum solutions while column #5 (the number of feasible solutions) shows the total number of feasible solutions. In Table 3, column #5 (the number of solutions) is the average number (count) of feasible solutions found with the associated average score.

Table 1: Experiment Criteria.

Provider	Off Day	Preference
#1	1st day of the month	No preference
#2	2nd day of the month	No preference
#3	3rd day of the month	Prefer to work during Week days

Table 2: Brute Force Search Results.

# of days	Brute Force			
	Score	Time	# of solutions	# of feasible solutions
4	1.2236	00:00:00.1216167	72	528
5	1.2178	00:00:01.0687332	360	6,480
6	1.1750	00:00:13.7340866	10,800	80,640
7	1.1981	00:02:46.5347595	75,600	907,200
8	1.1928	00:28:20.3932436	2,419,200	7,257,600
9	1.1928	02:45:10.2148520	16,450,560	74,592,000
10	1.1928	07:20:09.4689915	62,899,200	414,892,800

Table 3: Averaged Results of MCTS.

# of days	MCTS			
	# of play out	Score	Time	# of solutions
4	2,000	1.2119	0:00:01.88	4.4
	5,000	1.2003	0:00:04.69	2.8
	10,000	1.2236	0:00:08.92	2.4
5	2,000	1.2105	0:00:02.42	29.6
	5,000	1.2105	0:00:05.62	5.6
	10,000	1.2033	0:00:09.50	35.6
6	2,000	1.1711	0:00:02.09	8.8
	5,000	1.1750	0:00:05.12	5.0
	10,000	1.1750	0:00:09.53	9.8
7	2,000	1.1974	0:00:02.14	23.2
	5,000	1.1974	0:00:04.91	10.8
	10,000	1.1974	0:00:09.65	13.8
8	2,000	1.1928	0:00:01.84	195.0
	5,000	1.1928	0:00:04.48	4.8
	10,000	1.1928	0:00:08.59	16.6
9	2,000	1.1928	0:00:02.12	114.8
	5,000	1.1928	0:00:04.96	42.4
	10,000	1.1928	0:00:08.98	7.6
10	2,000	1.1928	0:00:01.98	207.2
	5,000	1.1928	0:00:04.85	48.6
	10,000	1.1928	0:00:09.31	441.0

5.1 Result Discussion

In Table 2 and Fig. 11, the time required for brute force search to completely evaluate the roster is an exponential function while MCTS' elapse times are predictable, relatively short, and mostly depends on the number of play out. The exceptions are the rosters with the total of four (4) and five (5) shifts per calendar month. This is due to the fact that MCTS algorithm has overheads (selection, expansion, simu-

Table 4: Elapse Time of BFS and MCTS.

# of days	Time			
	Brute Force	MCTS		
		2K	5K	10K
4	0:00:00.12	0:01.878	0:04.692	0:08.921
5	0:00:01.07	0:02.423	0:05.620	0:09.497
6	0:00:13.73	0:02.089	0:05.124	0:09.526
7	0:02:46.54	0:02.137	0:04.912	0:09.647
8	0:28:20.39	0:01.845	0:04.483	0:08.589
9	2:45:10.22	0:02.123	0:04.961	0:08.980
10	7:20:09.47	0:01.980	0:04.850	0:09.307

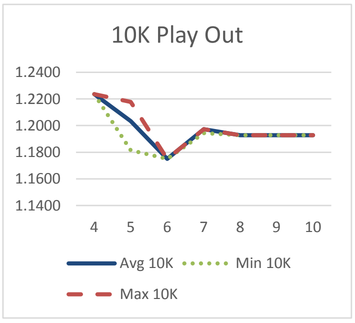


Figure 10: 10K Play Out Results.

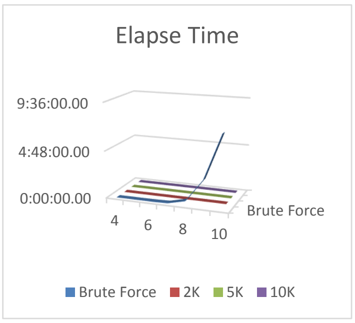


Figure 11: Elapse Time.

lation and back propagation) and the search space is not as huge as the other experiments, thus brute force search outperformed MCTS. From the experiment results as shown in Fig. 8, 9 and 10, we found that the solutions produced by MCTS gave us optimal solutions most of the time (or near optimum) and if given more time and resources, it would always produce optimal solutions. However, the time taken to produce the solutions were significantly shorter compared to brute force method.

6 FUTURE WORK

MCTS will not work well if the random sampling of the solutions always return the score of zero (0). In short, MCTS will not work well if there are only a few feasible solutions in the search space given a limited time span. Future work might explore ways of narrowing down the search of feasible solutions quickly and at the same time, optimizing the solutions.

7 CONCLUSION

SP-MCTS can be used to find global optimal solutions for the constraint based staff scheduling problem and uses significantly less time to find the solution than brute force search. The schedulers have the advantage of interrupting the search and still getting the best solutions out of it, and often these solutions are optimum (or near-optimum) solutions. SP-MCTS can be used to optimize other constraint based problems with the same methodology.

REFERENCES

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43.

Brucker, P., Burke, E., Curtois, T., Qu, R., and Vanden Berghe, G. (2010). A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, 16(4):559–573.

Burke, E. K., Li, J., and Qu, R. (2010). A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484 – 493.

Huang, H., Lin, W., Lin, Z., Hao, Z., and Lim, A. (2014). An evolutionary algorithm based on constraint set partitioning for nurse rostering problems. *Neural Computing and Applications*, pages 1–13.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer.

Matsumoto, S., Hirosue, N., Itonaga, K., Ueno, N., and Ishii, H. (2010). Monte-carlo tree search for a reentrant scheduling problem. In *Computers and Industrial Engineering (CIE), 2010 40th International Conference on*, pages 1–6.

Schadd, M., Winands, M., Herik, H., Chaslot, G.-B., and Uiterwijk, J. (2008). Single-player monte-carlo tree search. In Herik, H., Xu, X., Ma, Z., and Winands, M., editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg.

Sotskov, Y. and Shakhlevich, N. (1995). Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237 – 266.

Yang, F.-C. and Wu, W.-T. (2012). A genetic algorithm-based method for creating impartial work schedules for nurses. *International Journal of Electronic BusinessManagement*, 10(3):182.