Guided Monte Carlo Tree Search for Planning in Learned Environments

Jelle Van Eyck JELLE.VANEYCK@CS.KULEUVEN.BE
Department of Computer Science, KULeuven
Leuven, Belgium

Jan Ramon JAN.RAMON@CS.KULEUVEN.BE
Department of Computer Science, KULeuven
Leuven, Belgium

Fabian Guiza FABIAN.GUIZA@MED.KULEUVEN.BE
Laboratory of Intensive Care Medicine, KULeuven
Leuven, Belgium

Geert Meyfroidt GEERT.MEYFROIDT@MED.KULEUVEN.BE
Laboratory of Intensive Care Medicine, KULeuven
Leuven, Belgium

Maurice Bruynooghe MAURICE.BRUYNOOGHE.CS.KULEUVEN.BE
Department of Computer Science, KULeuven
Leuven, Belgium

Greet Van den Berghe GREET.VANDENBERGHE@MED.KULEUVEN.BE
Laboratory of Intensive Care Medicine, KULeuven
Leuven, Belgium

Editor: Cheng Soon Ong and Tu Bao Ho

Abstract

Monte Carlo tree search (MCTS) is a sampling and simulation based technique for searching in large search spaces containing both decision nodes and probabilistic events. This technique has recently become popular due to its successful application to games, e.g. Poker Van den Broeck et al. (2009) and Go Coulom (2006); Chaslot et al. (2006); Gelly and Silver (2012)). Such games have known rules and the alternation between self-moves and non-deterministic events or opponent moves can be used to prune uninteresting branches. In this paper we study a real-world setting where the processes in the domain have a high degree of uncertainty and the need for longer-term planning implies a sequence of (planning) decisions without any intermediate feedback. Fortunately, unlike the combinatorial complexity in strategic games, many real-world environments can be approximated by efficient algorithms on a short term. This paper proposes an MCTS variant using a new type of prior information based on estimating the effects of part of the world and explores its application to the problem of hospital planning, where machine learning algorithms can be used to predict the length of stay of patients for each of the different stages of their recovery.

Keywords: Monte Carlo Tree Search, learning for planning, hospital scheduling

1. Introduction

Monte Carlo Tree Search (MCTS) is a best-first search technique that was first introduced in the domain of game playing. In this context, MCTS uses stochastic playouts (simulations) and their associated outcomes to estimate the expected long-term result of a given move. Instead of spending an equal amount of time on each possible move, MCTS focuses on the most promising moves.

The success of MCTS in solving complex games suggests that it may also have a broader applicability. Real-world domains, however, often differ on a number of points with the typical strategic game environment. First, typical for games such as Go is that moves are played in turn: each move is immediately followed by an opponent's. This reveals for many actions their consequences early in the search. Pruning actions which turn out to be suboptimal early allows for focussing on the few remaining candidates which are competitive in the short run. For many problems, however, this does not hold. Consider, e.g., a problem where a number of decisions has to be made at once, without any intermediate feedback from the system. The larger the number of sequential decisions, the less feedback one receives and the less one can prune the search.

Second, unlike many strategic games with combinatorial search spaces, real-world environments often allow for approximate models which are accurate on a short term and can be searched efficiently. In such cases, it may be possible to combine an efficient and accurate local optimization with an MCTS based global search.

Third, while in games the rules are known, in a real-world environment the dynamics may be unknown. Often, an approximate model of the environment dynamics can be learned from training examples. However, one must take into account that the accuracy of the learned model may heavily influence the quality of a strategy based on simulating that model, such as MCTS.

In this paper, we study the application of MCTS for planning in such a real-world domain. Our contribution is two-fold. First, we propose a planning algorithm that combines a local model and global Monte Carlo tree search. Second, we apply this algorithm to the case of hospital planning where we perform an extensive experimental analysis.

The remainder of this paper is structured as follows. In section 2 we formulate the general problem, to which a solution is proposed in section 3. The details of the hospital planning problem are introduced in section 4 and experiments and results considering this case are discussed in section 5. Finally, in section 6, we conclude.

2. Problem formulation

We will consider environments where objects come in, are processed and then outputted. Generally speaking, the goal is to process as many objects as possible while completing every accepted object/task and ensuring all of these follow a high-quality trajectory. One example is a company accepting projects requested by customers: accepting more projects and realizing them yields a higher reward, but committing to too many projects carries the risk that one or more of the projects exceeds the capacity of the company, which may result in penalties. Another example, which we will study in detail in this paper, is a hospital accepting patients for surgery. Treating a larger number of patients yields a higher reward but failing to give patients the care they need may cause significant damage.

More formally, a partially observable Markov decision problem (POMDP) P is a tuple $P = (S, A, \tau, O, R)$ where S is a set of states, A is a set of actions, $\tau : S \times A \times S \rightarrow [0, 1]$ is a transition function (giving the probability $\tau(s_1, a, s_2)$ to reach state s_2 after executing action a in state s_1), $O : S \times A \rightarrow \mathcal{O}$ is an observation function mapping state-action pairs on observations in some space $\mathcal{O}$ and $R : S \times A \rightarrow \mathbb{R}$ is a reward function assigning to every state-action pair a reward. In our setting, we will denote the POMDP describing the behavior of objects (company project or patient) with $P_{obj} = (S_{obj}, A_{obj}, \tau_{obj}, O_{obj}, R_{obj})$.

We consider applications where quitting a previously accepted object/task is not acceptable. Therefore, in case of insufficient capacity to process the object additional expenses must be made. E.g. when a company commits to a critical project it must be followed up, even if this requires additional personnel or equipment. Similarly, once admitted, a patient must be treated even if this implies cancellation of other planned admissions.

In order to define such problems more formally, we will first introduce some notations. For a set X, X^n is the set of all n-tuples of elements of X. We use some common string-notations, in particular $X^* = \cup_{i \in \mathbb{N}} X^i$ and for $x \in X^n$, we denote with $x(i)$ the i-th element of x. We denote with $|x|$ the length of x. For $x \in X^*$ and $z \in X$, we denote with $freq(z, x)$ the number of i for which $x(i) = z$. For $x, y \in X^*$, we write $x \subseteq y$ if for every $z \in X$ we have $freq(z, x) \leq freq(z, y)$.

We define a Critical Project Planning Problem (CPPP) as a tuple $P_{CPPP} = (P_{obj}, S_S, S_T, L)$ where $S_S \subseteq S_{obj}$ is a set of beginning states, $S_T \subseteq S_{obj}$ is a set of terminal states and $L : A^*_{obj} \rightarrow \{0, 1\}$ is a function representing capacity limits. Given a tuple A of actions to perform on objects, $L(A)$ is 1 if it is possible to perform these actions simultaneously, else $L(A) = 0$. Naturally, L is anti-monotonic in the sense that if $A \subseteq A'$ and $L(A) = 0$ then $L(A') = 0$. To a CPPP there corresponds a global POMDP $P_g = (S_g, A_g, \tau_g, O_g, R_g)$ where $S_g = S^*_{obj}$ is the set of all tuples of object states of S_{obj} and likewise $A_g = A^*_{obj}$. Let $s \in S_g$ and $a \in A_g$. If $|s| = |a|$ and $L(a) = 1$, $\tau_g(s, a) = \prod_{i=1}^{|s|} \tau_{obj}(s(i), a(i))$, $R_g(s, a) = \sum_{i=1}^{|s|} R_{obj}(s, a)$ and $O_g(s, a)$ is a string of length $|s|$ such that for all i, $O_g(s, a)(i) = O(s(i), a(i))$. Else ($|s| \neq |a|$ or $L(a) = 0$), $\tau_g(s, a, s') = \delta(s, s')$ (the Kronecker delta), $O_g(s, a) = \emptyset$ and $R_g(s, a) = -\infty$.

We assume a CPPP starts in a state s which is a tuple with for each object a beginning state $s(i) \in S_S$, and that for $s \in S_T$, $\tau_{obj}(s, a, s') = \delta(s, s')$ and $R_{obj}(s, a) = 0$. A terminal state of P_g is an $s \in S_g$ with $s(i) \in S_T$ for all $i \leq |s|$. The idea of the resource limit function L is not that a disaster possibly occurs by receiving a reward $-\infty$, but that when a particular type of resources would become exhausted a more expensive emergency resource of the same type would be used (e.g. for company projects hiring temporary personnel). We assume that once an object is accepted (its state is no longer in S_S), it must always get the appropriate care (even if expensive) and hence assume that for any s and s' with $s \notin S_S$, $\tau_{obj}(s, a, s')$ does not depend on a.

3. Monte Carlo Tree Search

MCTS was first introduced in 2006 in three variants Coulom (2006); Kocsis and Szepesvári (2006); Chaslot et al. (2006). In general, MCTS is a technique for finding optimal decisions through a guided process of simulations. This process constructs an asymmetric tree in an incremental manner. Each node of the tree represents a state of the system and has a visit count associated with it, as well as an expected outcome.

A simulation starts at the root of tree and continues by sequentially selecting child nodes until a terminal node is reached. The selection strategy tries to balance exploitation and exploration of the tree, favoring scarcely visited nodes on the one hand, and nodes which are likely to yield better outcomes on the other. The most frequently used selection strategy is UCT Kocsis and Szepesvári (2006) (Upper bound Confidence for Trees), an adapted version of UCB Auer et al. (2002) (Upper Confidence Bounds).

Once a terminal node of the tree is selected, it is then expanded by generating and adding one or more of its children to the tree. Starting from these child nodes, simulation then continues until conclusion. The simulation strategy can either be fully at random, or based on prior information or heuristics. The latter strategy might exclude certain decisions, whereas the former might contain nonsensical decisions. The outcomes of these simulations are then backpropagated throughout the tree and the corresponding statistics are updated. This full process is illustrated in Fig. 1.

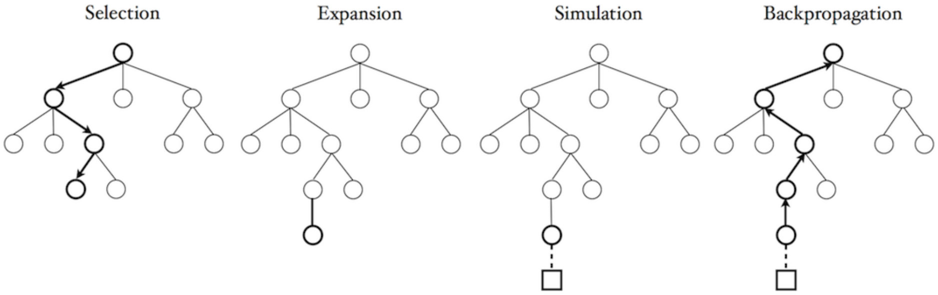

Figure 1: A general overview of the different steps of the MCTS algorithm.

While the basic algorithm has proven effective for a wide range of problems Browne et al. (2012), the full benefit of MCTS is typically not realized until this basic algorithm is adapted to suit the domain at hand. In the case of game playing, a lot of effort has gone into optimizing efficiency. Techniques such as progressive widening Coulom (2007) and progressive unpruning Chaslot et al. (2008) help control the size of the search space, while other techniques such as RAVE Gelly and Silver (2007) (Rapid Action Value Estimation) aim at actively reducing computing time.

In this paper too we use domain knowledge to guide MCTS, but our setting and use of action value estimates is significantly different from the one in Gelly and Silver (2007). In particular, we only have a model for the near future (rather than an estimate of the total expected future reward), our application is very different from the game of Go and (as a consequence) we can't use any of the optimisations such as "all moves as first" and $\alpha - \beta$-pruning used in Gelly and Silver (2007).

Another work related to ours is Runarsson et al. (2012), which applies MCTS to job shop scheduling. An important difference with our setting (which we'll explain in the next section) is that in our case the nature of jobs (sequence of states patients have to pass) and their duration (the length of stay of patients) is not known before scheduling them.

MCTS for CPPPs Here, we define two types of nodes: action nodes where decisions are made (e.g. calling patients, reserving resources) and probabilistic nodes where events happen according to some probability distribution (e.g. patient recovery, discharge). The MCTS algorithm iteratively performs a simulation of actions taken and probabilistic events happening. The probabilistic nodes do not always produce the same event (as these are uncertain at the time of planning), and iterating over them gives better estimates.

The MCTS algorithm consists of the following four steps, which are performed until time runs out:

1. **Selection**: During the selection step, a node is selected by traversing the MCTS tree from the root node onwards until some stopping criterion is satisfied. In general, this procedure only stops upon reaching a leaf node of the tree. However, internal probabilistic nodes can also end up being selected if their underlying probability distribution has not yet been sufficiently sampled.

   Which child node exactly is picked when traversing the tree depends on the type of nodes. In the case of probabilistic nodes, the probability of picking one is proportional to the probability of it occurring. In the case of action nodes, a node k is selected according to the UCT formula:

   $$k = argmax_{i \in I} \left(Q_{n_i} + C \sqrt{\frac{\ln vc_n}{vc_{n_i}}} \right).$$

   Here, Q_{n_i} is the expected reward of the i-th child node, vc_n and vc_{n_i} the number of visits to the parent and i-th child node respectively, and C a constant factor that represents the trade-off between exploration and exploitation.

2. **Expansion**: In this step, the previously selected node is expanded. This expansion step depends on the type of child nodes. In the case of probabilistic nodes, a single new child node is generated and added to the MCTS tree. In the case of action nodes, all actions which make sense (don't give a reward of $-\infty$) are added as new child nodes.

3. **Simulation**: Starting at each of the newly added nodes, simulations of the POMDP are performed. During this process, probabilistic events are sampled according to their assumed probability distribution and actions are sampled according to a fixed distribution. At the end, the total reward is recorded.

4. **Backpropagation**: The reward is then backpropagated up the tree to the root, starting at the newly added leaf node, according to the following procedure. If the current node is a probabilistic node, its parent's score becomes the mean of all of its children. If the current node is an action node, its parent's score becomes the maximum of all of its children. This process is repeated until the root node is reached.

Once time has run out, (one of) the action(s) at the root which was used often during the MCTS search is selected and performed in the actual real-world problem. When at a later time a new decision has to be made, a new MCTS search is performed starting from the new state.

Priors When domain knowledge is available, it is possible to initialize node statistics (visit count and total reward received) with a prior estimate of the expected reward (see e.g. Gelly and Silver (2007)), such that the sampling converges faster.

In many applications, getting an estimate of the long-term reward is hard, while estimating the expected short-term evolution is feasible. In our case, we assume that for $s \in S_{obj} \setminus S_S$ and $a \in A_{obj}$ such that $R_{obj}(s,a) > -\infty$, $\tau_{obj}(s,a,s')$ is independent of a. Hence, once objects are not in a beginning state any more, we can learn and simulate a model for their evolution independently of the actions taken. This allows us to predict which resources will be occupied in the near future and how much cost/reward the currently accepted objects will have generated.

In particular, suppose that, using a model, we can estimate (or simulate) the total reward R_{acc} generated by the set of objects already accepted up to some node n, assuming no further objects will be accepted in future. Let a node n of the MCTS tree have k children n_1, n_2, $\ldots$, n_k. We can write the total expected reward Q_{n_i} following the path through n_i as $Q_{n_i} = Q^{acc}_{n_i} + Q^{new}_{n_i}$ where $Q^{new}_{n_i}$ is the additional reward generated by accepting new objects after node n_i. While it is in many cases feasible to estimate $Q^{acc}_{n_i}$, it is often very hard to assess what additional reward the objects not yet accepted could bring. They may disappear (a project will be performed by another company, a patient dies or is treated at another hospital), and optimal timing of their acceptance is combinatorially hard. We will therefore adopt the following procedure to add our prior knowledge to the statistics of an MCTS node. Let for node n_i the number of visits to the node be vc_{n_i} and let the sum of the total rewards collected during these vc_{n_i} visits be $Q^{sim}_{n_i}$. Then, for our prior we use the same reward generated by not-yet-accepted objects for all children of n, i.e. we estimate $\hat{Q}^{new}_{child(n)} = \frac{1}{k}\sum_{i=1}^{k}\left(Q^{sim}_{n_i}/vc_{n_i} - \hat{Q}^{acc}_{n_i}\right)$ where $\hat{Q}^{acc}_{n_i}$ is the prior knowledge based estimation of $Q^{acc}_{n_i}$. As prior for the total reward of node n_i we use $Q^{prior}_{n_i} = Q^{acc}_{n_i} + \hat{Q}^{new}_{child(n)}$. Then, as is common, we consider as estimated reward for node n_i the combination of prior and samples $\hat{Q}_{n_i} = (C_{prior}Q^{prior}_{n_i} + Q^{sim}_{n_i})/(C_{prior} + vc_{n_i})$ where C_{prior} is a constant giving the relative importance of the prior.

4. Hospital planning

In a hospital, scheduling patients for elective cardiac surgery is a challenging task that involves the assignment of several of the hospital's resources. In order to guarantee an optimal throughput of patients, it is essential that these resources are used in an optimal manner.

Once patients are admitted to the hospital, all resources for treatment and stay must be available. Some resources should be reserved in advance, and hence it is necessary to make a schedule. Unlike standard job shop scheduling, this resource reservation is not a very combinatorial problem. Here, the main challenge is the fact that it is unknown how patients, as well as the availability of resources, will evolve over time.

First, we will discuss our implementation of a virtual cardiac surgery unit in Section 4.1. Next, we will discuss length of stay prediction, which is used to estimate the distribution of probabilistic events, in Section 4.2.

4.1. Virtual cardiac surgical unit

We have built a virtual cardiac surgical unit which allows us to simulate the dynamics of its real counterpart in a realistic (albeit somewhat simplified) manner. A patient can be in one of the states $S_{pat} = \{$`healthy`, `waiting_list`, `called`, `ward`, `surgery`, `ICU`, `discharged`, `deceased`$\}$. Here, being `healthy` or being on the waiting list (`waiting_list`) are beginning states, and being discharged after treatment (`discharged`) or having deceased (`deceased`) are terminal states. The possible transitions are illustrated in Fig. 2. Initially, when a formerly healthy patient requires elective cardiac surgery, he is added to a waiting list. Each week, a number of patients is selected from this list and scheduled to undergo surgery the following week. These patients are notified of the schedule and the necessary resources are reserved. Upon their arrival patients are admitted to the ward, unless no bed is available, in which case they are sent back home. After undergoing surgery, recovering patients require a bed at the intensive care unit (ICU). However, if no such bed can be guaranteed, surgery is postponed and the patient is sent home. Once a patient has sufficiently recovered, he is transferred to the ward. A patient usually remains here until he can be discharged. However, when complications arise, the patient might require additional ICU care followed by additional care at the ward. In some cases, when complications are particularly severe, the patient might not survive surgery or the following stay.

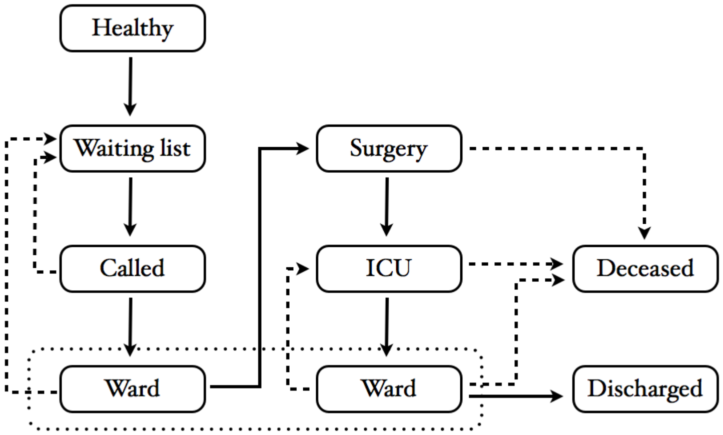

Figure 2: Flowchart summarizing the different possible patient flows.

In practice, both ICU and ward can ’overflow’ due to several reasons. There is not always a completely deterministic procedure to decide on priority. For instance, if surgery is initiated and a patient from the ward must go back to the ICU, the ICU accepts one patient more than its normal capacity for cardiac surgery. On the other hand, if first the patient moves from the ward to the ICU and then a scheduled patient arrives, he is sent back home. To avoid such non-deterministic behavior in our virtual hospital the ICU always accepts patients, but a significant additional cost is incurred when the ICU overflows. In particular, patients are not sent home but get an expensive ’overflow’ bed. Similarly, a full ward may lead to arriving patients being sent home, patients staying at ICU rather than moving or the ward accepting more patients than its normal capacity. Again, in our simulation we model this with expensive overflow beds.

The actions one can perform on a patient in the waiting list are `no-action` and `call`. For a patient needing a bed in the ICU, actions `in_icu` and `overflow_icu` are possible. For a patient needing a bed on the ward, actions are `on_ward` and `overflow_ward`. Clearly, for patients which are `healthy`, `discharged` or `deceased` only `no-action` is possible.

The two major categories of costs in our virtual hospital are beds not being used for cardiac surgery patients (in practice they won't be empty, but such beds slow down the cardiac surgery program), and patients who must be put on overflow rather than regular beds. In our simulation, for a bed not used for a cardiac surgery patient we use a cost (negative reward) of 0.16 per day. For a patient in an overflow bed, we use a cost of 5 per day.

4.2. Predicting LOS

Previously, we have shown Meyfroidt et al. (2011); Ramon et al. (2007) that it is feasible to predict a patient's length of stay (LOS) after cardiac surgery accurately using data (physiological information, laboratory results, administered treatments, ...) collected during the first few hours of his stay in the ICU. Using these data sources, a Gaussian Process (GP) model Rasmussen and Williams (2005) was learned. The accuracy of the resulting model was validated against that of a general scoring model EuroSCORE Nashef et al. (1999), nurses and physicians. The GP model was shown to be able to outperform nurses and EuroSCORE on this specific prediction task, but was found to be equivalent to physicians. However, in contrast to physicians, who are often too busy, the model is capable of making predictions for all patients at an earlier point in time. It seems plausible that one can also predict LOS at the ward (where patients go after their ICU stay) based on the data collected during the ICU stay, though we are not aware of such a study. Several studies have considered the dependency of length of stay and survival Nashef et al. (1999); Toumpoulis et al. (2005) on pre-admission examination results. In conclusion, it is realistic to assume that one can build models predicting at any point in time reasonably accurately the stay and progress of a patient in the near future.

Unfortunately it was not possible to directly incorporate LOS predictors in this research due to an insufficient amount of patient data. More precisely, information regarding physiological parameters and administered treatments was not available for patients during their ward stay. In order to overcome this, we use simulated LOS predictors that have the same properties as regular predictors.

Fig. 3 and 4 respectively show the distribution of the length of stay in ICU and ward for a population of virtual patients of the university hospitals (see Section 5 for details). These distributions can be approximated quite well with Poisson distributions. This motivates us to model LOS predictors as functions outputting Poisson distributions. If no information is available, they output the Poisson distribution fitted to the complete population. If a predictor is more accurate, it outputs for every patient a more narrow Poisson distribution.

More precisely, we assume that in order to go through a particular stage of his stay (ICU or ward), a patient must pass a number of steps z. The average number of steps taken per unit of time is $1/T_s$. In order to simulate the progress of a patient (and the assessment which

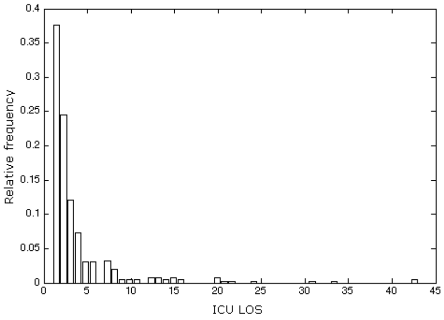

Figure 3: Histogram of ICU LOS.

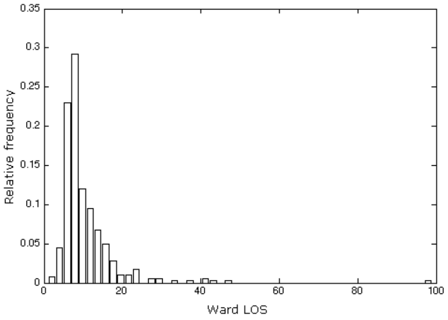

Figure 4: Histogram of ward LOS.

can be made during the stay of a patient), a predictor performs the following steps. First, from the actually recorded LOS L (which is hidden to (not yet known by) the planner until discharge) the predictor samples a number of steps z the patient had to progress during that stay according to $P(z|L) = \frac{e^{-L/T_s}(L/T_s)^z}{z!}$ and randomly distributes z events of "taking one step" over the time interval of length L. Then, whenever a prediction is needed, the predictor looks (in the list of step events generated above) how many steps z' the patient still has to take between now and his moving away from the current hospital department (ICU or ward), and returns a probability distribution $P(L'|z')$ indicating the probability that taking z' steps takes L' time.

Using this way of modeling has several advantages. First, the prediction may be initially far off but will evolve to the correct LOS as time progresses (which corresponds to reality). Second, the predictor returns a Poisson distribution as its prediction, which approximates well the uncertainty of a real LOS predictor. Third, we can vary the parameter T_s. Smaller T_s values will give more accurate predictors. We will call T_s the accuracy of the predictor.

We can then compute the probability distribution over the total expected reward for every day d caused by the already accepted patients. For this, we use a dynamic programming strategy computing for every d, s_{ICU} and s_{ward} the probability that on day d exactly s_{ICU} ICU beds and s_{ward} ward beds will be needed. Summing over all costs, weighted by the probability of occurring them, gives the desired Q_n^{acc} estimate.

5. Experiments

In this section we discuss the various experiments. The concrete experimental setup is introduced in section 5.1. Results are presented in section 5.2 and discussed in section 5.3.

5.1. Experimental setup

A virtual patient pool was created, consisting of 400 virtual adult cardiac surgery patients. The properties of these virtual patients were selected from an anonimized administrative database of all patients undergoing elective cardiac surgery at the university hospitals between October 2010 and November 2011, and for whom sufficient data was available.

Table 1: Results for fixed planning, where each day a fixed number of patients is admitted.

Fixed planning					
$\#admitted$	c_{ICU}	c_{ward}	c_{unused}	c_{tot}	$t_{run}(s)$
1	0	0	29360	4697	72
2	0	0	12191	1950	35
3	0	0	6476	1036	24
4	0	0	3607	577	18
5	0	35	1897	478	15
6	7	351	801	1918	13

Being able to accurately determine the flow of each individual virtual patient is important for this study to be realistic. In order to do this, the ICU LOS and ward LOS of each of the selected patients was retrieved. In our dataset, the median stay of a patient at the ICU was 2 days. At the ward, the mean LOS counted 8 days. 385 of the 400 selected patients had a stay without complications: after surgery they went straight from ICU to ward and were discharged afterwards. 7 patients, on the other hand, required additional ICU care after their stay at the ward. 8 patients died due to complications.

This information allows us to simulate a cardiac surgical unit as described in Section 4.1. The virtual unit we use in our experiments consists of 22 ICU beds, 40 ward beds and 3 operating rooms. 3 surgeons perform surgery up to 2 times a day. These numbers (except for a simplification regarding surgeons) are identical to the clinical practice at the university hospitals. In each experiment, the hospital must admit and treat all patients in the database. Every week starts with planning on each day the number of patients to admit. After these 5 action nodes, the 5 days of the week with surgery and the 2 weekend days without surgery are simulated. This cycle is repeated until all patients are discharged.

Table 2: MCTS results for varying number of iterations, without prior knowledge, using predictors with $T_s = 1$.

No prior, $T_s = 1$					
$iter$	c_{ICU}	c_{ward}	c_{unused}	c_{tot}	$t_{run}(s)$
300	0 ± 0	21 ± 30	2940 ± 49	577 ± 111	344 ± 15
1000	0 ± 0	29 ± 14	2125 ± 65	485 ± 94	1290 ± 133
3000	0 ± 0	37 ± 22	1766 ± 369	469 ± 122	3627 ± 256
10000	0 ± 0	43 ± 19	1345 ± 51	432 ± 134	11532 ± 1072
30000	0 ± 0	37 ± 19	1000 ± 60	345 ± 142	34665 ± 3344

In our simulation experiments, we focus on the following three questions:

Table 3: MCTS results for varying accuracies, without prior knowledge, using 10000 iterations per decision.

No prior, *iter* = 10000					
T_s	c_{ICU}	c_{ward}	c_{unused}	c_{tot}	$t_{run}(s)$
0.3	0 ± 0	5 ± 4	1876 ± 39	325 ± 53	14898 ± 309
0.7	0 ± 0	17 ± 9	1614 ± 41	344 ± 74	13677 ± 1357
1	0 ± 0	43 ± 19	1345 ± 51	432 ± 134	11532 ± 1072
2	3 ± 3	20 ± 16	2358 ± 28	490 ± 82	10128 ± 290
3	0 ± 0	2 ± 5	3307 ± 94	540 ± 76	11106 ± 593
5	0 ± 0	0 ± 0	6129 ± 64	981 ± 64	14056 ± 1378
7	0 ± 0	0 ± 0	8151 ± 73	1304 ± 73	17559 ± 1286

Q1 What is the impact of using MCTS instead of a fixed planning such as currently used in our university hospitals?

Q2 When using MCTS, what is the effect of the accuracy of LOS predictors?

Q3 What is the effect of incorporating domain knowledge as explained in Section 3?

Where applicable, we report average results and standard deviations over ten repetitions of the concerned experiment. We report the number of patient-days where a patient was in an overflow bed in the ICU c_{ICU}, the number of patient-days where a patient was in an overflow bed on the ward c_{ward}, the number of bed-days where the bed was not used for a cardiac surgery patient c_{unused}, total cost c_{tot} and runtime t_{run}.

Every experiment has the following parameters:

- The number of iterations of the MTCS search each time an action must be made.
- The accuracy of the predictor (the value of the parameter T_s as explained above).
- Whether a domain knowledge based prior has been provided or not.

As an exception, the baseline experiments where a fixed number of patients are admitted every day don't have these parameters.

5.2. Results

Currently, elective cardiac surgery is planned in many hospitals simply by assuming that a fixed number of ICU beds will become available for cardiac surgery patients each day, and by planning that fixed number of surgeries. Table 1 shows the result of applying this fixed planning strategy for k patients with $k = 1 \ldots 6$. Here, these results will be used as a baseline.

Table 2 shows for a predictor of constant accuracy 1 the total cost as a function of the number of MCTS iterations.

Table 3 shows for a fixed number of 10000 MCTS iterations the results as a function of the predictor accuracy.

Table 4 compares MCTS using a prior with the same experiment not using a prior and using an equal amount of iterations and the same experiment using approximately the same amount of time. These experiments are performed for varying accuracies.

Table 5 compares MCTS with and without using priors in the expansion phase, and with and without priors in the simulation phase. Again, these experiments are performed for varying accuracies. Furthermore, the number of iterations was chosen in such a way that it takes on average 7200 seconds to perform each of the experiments.

Table 4: Comparison of MCTS with and without prior knowledge.

	Prior, *iter* = 1000		No prior, *iter* = 1000		No prior, comparable timing	
T_s	c_{tot}	$t_{run}(s)$	c_{tot}	$t_{run}(s)$	c_{tot}	$t_{run}(s)$
0.1	283 ± 32	13624 ± 862	375 ± 57	1627 ± 85	294 ± 49	13388 ± 225
0.3	317 ± 38	12572 ± 365	393 ± 32	1636 ± 174	335 ± 52	12284 ± 209
0.7	372 ± 30	12370 ± 945	435 ± 32	1390 ± 97	383 ± 27	12216 ± 334
1	388 ± 68	10812 ± 1058	504 ± 40	1087 ± 36	426 ± 139	10615 ± 215
3	491 ± 33	14211 ± 981	522 ± 56	931 ± 86	489 ± 48	13885 ± 519
7	987 ± 87	25289 ± 1605	1147 ± 98	1391 ± 56	1123 ± 47	25074 ± 1568

Table 5: Comparison of different combinations of using prior information (P) and no prior information (NP) during the expansion and simulation phases respectively. Each experiment takes on average 7200 seconds to perform.

	NP-NP		P-P		NP-P		P-NP	
T_s	c_{tot}	*iter*	c_{tot}	*iter*	c_{tot}	*iter*	c_{tot}	*iter*
0.3	333 ± 23	5500	291 ± 24	600	466 ± 54	550	269 ± 67	5500
1	443 ± 75	6500	399 ± 26	695	463 ± 42	650	414 ± 175	7200
3	489 ± 37	7500	475 ± 42	575	699 ± 62	550	480 ± 23	9500
10	1269 ± 66	3700	1015 ± 86	190	978 ± 88	180	1307 ± 116	3700

5.3. Discussion

Comparing table 1 and table 2, one can see that MCTS outperforms a fixed strategy even for a moderate amount of iterations. We can therefore answer Q1 that MCTS planning would be better than current practice assuming that the simulations are sufficiently realistic and the predictors have an accuracy similar to the accuracy of our simulated predictors. We have experimented with several variations in the cost schemes, and the obtained results are along the same lines. From Ramon et al. (2007) it appears that it is possible to learn sufficiently good predictors. In a future study we hope to assess our method in a real-world setting.

Concerning question Q2, as can be observed from table 3, the quality of the LOS predictors significantly influences the performance. Even moderately weak predictors allow for MCTS results competitive to the fixed strategy.

Concerning question Q3, it can be inferred from table 4 that using MCTS with domain knowledge takes more time per iteration (about a factor 11 more), but outperforms standard MCTS both for the same number of iterations and the same runtime budget. This is a promising result especially as significant runtime optimization is possible in our dynamic programming implementation. Furthermore, as can be seen in table 5, prior information seems to be more important during the tree expansion phase than the simulation phase.

6. Conclusions

The choice of a strategy for planning staff and patient admission in the context of elective cardiac surgery patients can have a significant influence on the total cost incurred. As a first step towards a better understanding of the impact, we have compared several realistic options in a simulation of a retrospective patient population.

In particular, we have developed a new MCTS variant suited for this and similar problems. More specifically, we expect our approach is applicable to problems where the allocation of resources is not a combinatorial problem but the evolution of running projects/patients is highly uncertain. In such case, using MCTS shows to be a decent approach, and using domain knowledge can help.

In future work, we intend to further refine the integration of prior information, to optimize our computation schemes to compute the priors, and to consider alternative simplified priors which may have a better cost-benefit ratio. Also, motivated by our application results, in future work we intend to build and validate a refined planning system based on complete patient data integrated over all relevant hospital departments, and to integrate our algorithm and models in a practical planning tool.

Acknowledgments

Jelle Van Eyck is supported by FWO grant G.0310.08. Jan Ramon is supported by ERC StG 240186 'MiGraNT'. Geert Meyfroidt is funded by the Research Foundation - Flanders (FWO) (Senior Clinical Investigator, 1843113N). Greet Van den Berghe, through the KULeuven, receives longterm research financing via the Flemish government Methusalem-program. Greet Van den Berghe also holds an ERC Advanced grant (AdvG-2012-321670) from the Ideas Program of the EU FP7.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. ISSN 0885-6125.

Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon

Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 03/2012 2012.

Guillaume Maurice Jean-Bernard Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk, and H. Jaap van den Herik. Monte-Carlo Strategies for Computer Go. In *Proc. BeNeLux Conf. Artif. Intell.*, pages 83–91, 2006.

Guillaume Maurice Jean-Bernard Chaslot, Mark H. M. Winands, H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. *New Math. Nat. Comput.*, 4(3):343–357, 2008. ISSN 1793-0057.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, CG'06, pages 72–83. Springer-Verlag, 2006.

Rémi Coulom. Computing Elo Ratings of Move Patterns in the Game of Go. *Int. Comp. Games Assoc. J.*, 30(4):198–208, 2007.

Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 273–280. ACM, 2007.

Sylvain Gelly and David Silver. The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions. *Communications- ACM*, 55(3):106–113, 2012.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Euro. Conf. Mach. Learn.*, pages 282–293, Berlin, Germany, 2006. Springer.

Geert Meyfroidt, Fabián Güiza, Dominiek Cottem, Wilfried De Becker, Kristien Van Loon, Jean-Marie Aerts, Daniel Berckmans, Jan Ramon, Maurice Bruynooghe, and Greet Vanden Berghe. Computerized prediction of intensive care unit discharge after cardiac surgery: development and validation of a gaussian processes model. *BMC Med. Inf. & Decision Making*, 11:64, 2011.

S.A. Nashef, F. Roques, P. Michel, E. Gauducheau, S. Lemeshow, and R. Salamon. European system for cardiac operative risk evaluation (euroscore). *Eur J Cardiothorac Surg*, 16(1):9–13, july 1999.

Jan Ramon, Daan Fierens, Fabián Güiza, Geert Meyfroidt, Hendrik Blockeel, Maurice Bruynooghe, and Greet Vanden Berghe. Mining data from intensive care patients. *Advanced Engineering Informatics*, 21(3):243–256, 2007.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Thomas Philip Runarsson, Marc Schoenauer, and Michèle Sebag. Pilot, rollout and monte carlo tree search methods for job shop scheduling. *CoRR*, abs/1210.0374, 2012.

I.K. Toumpoulis, C.E. Anagnostopoulos, D.G. Swistel, and J.J. DeRose. Does euroscore predict length of stay and specific postoperative complications after cardiac surgery? *Eur J Cardiothorac Surg*, 27(1):128–133, january 2005.

Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo tree search in poker using expected reward distributions. In *Lecture Notes in Computer Science,*, pages 367–381. Springer, November 2009.