Contents

Chapter 1

Approach

In this chapter I will give the set of definitions that define our approach. Some of these definitions make assumptions about the problem. If this is the case, the assumption, which consequently holds for the whole approach, will be given. Furthermore, examples of how the approach would work in certain scenarios will also be provided.
The approach is meant to solve a online planning-task. This means, that both the calculation and the execution of the plan are concurrent. Since in these kind of task we do not expect to be able to solve the whole plan until we have to start executing the plan, we are not interested in finding the best possible plan, but instead we are interested in finding the top-level action where we the expected path cost is minimized. Top-level action describes the first possible action we can take given our current position. Since we want to minimize the expected value it makes sense to consider a probability distribution over the possible path lengths instead of a single heuristic value. With this we can then choose the top-level action with the lowest expected value. The details of how this is done will be described in more detail later in the chapter.

1.1 Problem Properties

Even though the approach is meant to work for general problem solving we need to make some assumptions about the problem. These assumptions are needed either to properly define what we are doing or in order to properly evaluate the performance of the approach. In this section the general assumptions about the problem a presented and a short insight about the reason behind them is given. The need for some of the assumptions is clear immediately, while the need for others only becomes clear with the knowledge of the Definitions where the assumptions are used.

Assumption 1.

In each step the algorithm has enough time to expand exactly n nodes. Where n in some number that is fixed for the planning task.

This assumption greatly simplifies how we can talk about the problem. While in practice we usually have a certain amount of time to calculate which top-level action, it is not

easy to talk about this theoretically since it depends on to many factors. These factors include how the expansion is implemented and on which machine the program is running. However we are not interested in these information, since we want to have a general approach that is independent of the implementation and the machine t is running on.
An other advantage of this assumption is, that we can ignore when different nodes take varying time to expand. There are several reasons why this might happen, for example the calculation of the heuristic function might take varying time to calculate. Since we are ignoring the time it takes to expand a node, we can focus on the task of deciding which expansions of nodes give the most information instead of the task of looking at the information per time. Furthermore this again leads to a more general approach, since it does not depend on the specific heuristic that is used or how it is calculated.

Assumption 2.

There is no outside interference to the problem.

This assumption makes sure that the problem does not change while the search is running. The approach is not meant for this kind of problem and thus it might not work well if this happens. Furthermore there is no strategy defined how to deal with such behavior. Outside interference includes different things one of them would be that an other agent is working in the problem and thus actively changing it. But it also includes random events, that either the problem is changed randomly or the outcome of an action is randomly.
This assumption makes sure that a certain run of the approach is deterministic. This means if the approach is executed several times and the calculated probability distributions are calculated identically each time, the outcome of the algorithm is always the same. This makes it also much easier to evaluate the approach since there is no element of uncertainty.
Now that we have discussed the general assumptions about the problem we can begin to define the necessary things for our approach.

1.2 Decision Strategy

In this section the Strategy, how the system makes a decision based on a given Search-tree will be discussed. For this purpose we also have to define the form which the information we get from each node takes. For this we differentiate between leaf nodes, that are at the edge of the search space and inner nodes, which have already been expended and thus their child nodes are known.
Most Planning task usually take a single heuristic value. However in our approach we are not interested in such a single heuristic value. The reason for this is that we want to consider all possible path lengths from the current node to the goal. Only if we consider each possibility we can choose the path where the expected path cost is minimized since otherwise we have no measurement of how likely it is that the heuristic value can truly be achieved, and by how much it differentiates if this is not the case. This leads to the definition of the probability distribution, that is given in the leaves nodes of the search tree.

The heuristic function of the leaf nodes is defined as a finite set of pairs. The first value of the pair is the probability, that a certain value is the true path length to the goal. The second value is the corresponding value.

Definition 1. Probability distribution of leaves

$P_h(s) =$ The set of all pairs $(P(h^*(s) = n), n)$, for all n with $n < M \in \mathbb{N}$ for some upper bound M and $\sum^n P(h^*(s) = n) = 1$

M is the upper bound on the longest possible path length from any node in the problem to the goal.

$P(h^*(s) = n)$ is the estimated probability that the shortest possible path from the node s has length n.

What should be noted about this definition is the possibility, that some of the probabilities $P(h^* = n)$ are 0. In this case we can omit the pair from the set. This will not influence any of the later calculations, since any pair where the probability is zero has no infect on the probability at the parent node. Furthermore, one should note, that $h^*(s)$ is usually just one number. Thus $P(h^*(s) = n)$ should be 1 for this number and 0 everywhere else. However we do not always know this number. Thus we can only guess what the probability for certain numbers are. In order to not create unnecessary many notations $P(h^*(s) = n)$ is the probability that we assume the shortest path has to the goal.
Furthermore one should note that the approach does not specify how to obtain this probability distribution. Possible approaches would be for example to use a machine learning approach that learned how such a distribution might look like on similar problems. An other approach could be to use several inadmissible and admissible heuristics and interpolate between them in order to obtain the probability distribution.
Independently of how we obtained it the probability distribution should have the following property.

Assumption 3.

The probability distribution in the leaf node is correct. In particular, the probability of the true shortest distance to the goal is greater 0.

This property makes sure that what we are doing makes actually sense. If the calculated probability that the shortest path to the goal is 0 for some n while this is the actual value there must be something wrong in the strategy. Furthermore this property leads us to some nice properties later on, that help us to argue why the approach is reasonable.
Now that we have defined how the probability distribution in the leaves look like we can start to define how we want the probabilities distributions of inner nodes and in particular the nodes that are next to the current node. The probability distribution of an inner node should represent the probabilities of its child node in such a way that we consider all possible values it can take from this node. This means that the value that corresponds to the lowest value that is the highest value in any of the nodes should be the highest value and the other values should be the probabilities over all child nodes that this is the

shortest path to the goal. One way of getting the probability distribution of a parent node from the probabilities of the child nodes in a continues setting is the Cserna Update.

Definition 2. General probability distribution of inner nodes

Given a node s with two child nodes s_α, s_β, where the heuristic function of the child nodes has already been calculated. The probability distribution of the node s is then calculated from the probability distribution of s_α, s_β.

$P_h(s,d) := \sum_{d_\alpha, d_\beta : min(d_\alpha, d_\beta)=d} P_h(h^*(s_\alpha) = d_\alpha \cup h^*(s_\beta) = d_\beta)$

Since there is no general way to calculate $P(h^*(s_\alpha) = d_\alpha \cup h^*(s_\beta) = d_\beta)$ we need to constrain our problem to those cases where we can calculate this. One particular case where this would be possible would be if the probability distributions would be independent, since then it would be possible to calculate this by multiplying the probability distributions.

Assumption 4.

The probability distributions on different branches of the search tree are independent from each other.

This assumption makes sure that we can predict the outcome of combining two different probability distributions of child nodes without the need to take into account how one of them influences the other. This leads to the definition of how we can update the probability distribution of an inner node given the probability distributions of its child nodes.

Definition 3. Probability distribution of inner nodes with independent child nodes

Given a node s with two child nodes s_α, s_β, where the heuristic function of the child nodes has already been calculated. The probability distribution of the node s is then calculated from the probability distribution of s_α, s_β.

$P_h(s,d) := \sum_{d_1, d_2, ..., d_n : min(d_1, d_2, ..., d_n)=d} P_h(s_1, d_1) \cdot P_h(s_2, d_2) \cdot ... \cdot P_h(s_n, d_n)$

This definition allows us to calculate the probability distribution of a parent node. The following examples shows how the calculation would work for the case where the node has exactly two child nodes.

Example 1. *Calculating Cserna-Update with two child nodes:*
Given the following
$P_h(s_\alpha, d) = (10, 0.5), (20, 0.5)$,
$P_h(s_\beta, d) = (14, 0.5), (16, 0.5)$
The probability distribution of the parent node is computed:
(10,0.5) x (14,0.5) : d=10, s = 0.25

(10,0.5) x (16,0.5) : d=10, s = 0.25
(20,0.5) x (14,0.5) : d=14, s = 0.25
(20,0.5) x (16,0.5) : d=16, s = 0.25
Add those with the same d values up
=>Result: {(10,0.5),(14,0.25),(16,0.25)}

As one can see on the example the Cserna-Update is relatively straight forward for two nodes. However in many cases a node can have more than two child nodes. One way would be to just apply the formula for several nodes. An alternative way would be to recursively apply the formula for two child nodes until all child nodes have been added. In fact both of these methods are equivalent as shown in the following proof. In particular what we want to proof is the following:

Lemma 1. *Let*
$P_h(s',d') = \sum_{d_1,d_2,...,d_n:min(d_1,d_2,...,d_{n-1})=d} P(s_1,d_1) * P_h(s_2,d_2) * ... * P_h(s_{n-1},d_{n-1})$
Then:
$P_h(s,d) = \sum_{d',d_n:min(d',d_n)=d} P_h(s',d') * P_h(s_n,d_n)$

Proof. Proof via Induction:

Base Case n=2:
$P_h(s',d') * P_h(s_2,d_2) = \sum_{d_1,d_2:min(d_1,d_2)=d} P_h(s_1,d_1) * P_h(s_2,d_2) = P_h(s,d)$

Induction step: n-1->n:
$\sum_{d',d_n:min(d',d_n)=d} P_h(s',d') * P_h(s_n,d_n)$
$= \sum_{min(d_1,...,d_{n-1}),d_n:min(min(d_1,...,d_{n-1}),d_n)=d} (P_h(s_1,d_1) * ... * P_h(s_{n-1},d_{n-1})) * P_h(s_n,d_n)$
$= \sum_{d_1,...,d_{n-1},d_n:min(d_1,...,d_{n-1},d_n)=d} P_h(s_1,d_1) * ... * P_h(s_{n-1},d_{n-1}) * P_h(s_n,d_n)$
$= P_h(s,d)$

□

In practice the second version, that is to apply the formula iteratively might be preferable, since it is easier to compute. The computational time for the first version would be given by n^m ,with n= number of different probabilities in each distribution, m = number of nodes. The computational time for the second version on the other hand would be $m * 2^n$, which is lower for large n.

Example 2. *Calculating Cserna Update, more than two(three) branches:*
Given the following probability distributions
$P_h(s_\alpha,d) = (10,0.5),(20,0.5)$,
$P_h(s_\beta,d) = (14,0.5),(16,0.5)$,
$P_h(s_\gamma,d) = (12,0.5),(18,0.5)$
The probability distribution of the parent node is computed by applying the Formula to all nodes at once:

(10,0.5) x (12,0.5) x (14,0.5) : d=10, s = 0.125
(10,0.5) x (12,0.5) x (16,0.5) : d=10, s = 0.125
(10,0.5) x (18,0.5) x (14,0.5) : d=10, s = 0.125
(10,0.5) x (18,0.5) x (16,0.5) : d=10, s = 0.125
(20,0.5) x (12,0.5) x (14,0.5) : d=12, s = 0.125
(20,0.5) x (12,0.5) x (16,0.5) : d=12, s = 0.125
(20,0.5) x (18,0.5) x (14,0.5) : d=14, s = 0.125
(20,0.5) x (18,0.5) x (16,0.5) : d=16, s = 0.125
Add those with the same d values up
=>Result: {(10,0.5),(12,0.25),(14,0.125),(16,0.125)}

Alternative calculation (applying the Formula iteratively to the result of Example 1):
{(10,0.5),(14,0.25),(16,0.25)} x (12,0.5),(18,0.5):
(10,0.5) x (12,0.5) : d=10, s = 0.25
(10,0.5) x (18,0.5) : d=10, s = 0.25
(14,0.25) x (12,0.5) : d=12, s = 0.125
(14,0.25) x (18,0.5) : d=14, s = 0.125
(16,0.25) x (12,0.5) : d=12, s = 0.125
(16,0.25) x (18,0.5) : d=16, s = 0.125
Add those with the same d values up
=>Result: {(10,0.5),(12,0.25),(14,0.125),(16,0.125)}

In this example the effort to calculate the probability is greater for the second version. The reason for this is that the amount of nodes is still relatively small in comparison with the amount of possible values. If the amount of nodes grows, the effort grows much more for the first version, thus the second version becomes easier for a large number of nodes. In order to acquire the probability distribution in the nodes that follow the top-level action one can do either of the following two things. Either one applies the Cserna-Update bottom up until one reaches the parent node or one calculates it directly from all leaves nodes that are beneath the top-level action. The result of both these strategies is the same. In the following i will define both these strategies mathematically and proof that the result is equivalent.

Lemma 2. *Let*
$P(s,d)_1 = \sum_{d_1,d_2,...,d_m:min(d_1,d_2,...,d_m)=d} P(s_1,d_1) * P(s_2,d_2) * ... * P(s_m,d_m)$
$P(s,d)_2 = \sum_{d_{m+1},d_{m+2},...,d_n:min(d_{m+1},d_{m+2},...,d_n)=d} P(s_{m+1},d_{m+1}) * P(s_{m+2},d_{m+2}) * ... * P(s_n,d_n)$
Then:
P(s,d)$= \sum_{d(1),d(2):min(d(1),d(2))=d} P(s,d)_1 * P(s,d)_2$

Proof. Proof via Induction:

Base Case n=2:
$P_h(s,d)_1 = P_h(s_1,d_1)$

$P_h(s,d)_2 = P_h(s_2,d_2)$
$\sum_{d(1),d(2):min(d(1),d(2))=d} P_h(s,d)_1 * P_h(s,d)_2 = \sum_{d_1,d_2:min(d_1,d_2)=d} P_h(s_1,d_1) * P_h(s_2,d_2) = P_h(s,d)$

Induction step: m,n-m-> n:

$$\sum_{d(1),d(2):min(d(1),d(2))=d} P_h(s,d)_1 * P_h(s,d)_2$$
$$= \sum_{d_1,d_2,\ldots,d_m,d_{m+1},d_{m+2},\ldots,d_n:min(min(d_1,d_2,\ldots,d_m),min(d_{m+1},d_{m+2},\ldots,d_n))=d} P_h(s_1,d_1) * P_h(s_2,d_2) * \ldots * P_h(s_m,d_m) * P_h(s_{m+1},d_{m+1}) * P_h(s_{m+2},d_{m+2}) * \ldots * P_h(s_n,d_n)$$
$$= \sum_{d_1,d_2,d_n:min(d_1,\ldots,d_n)=d} (P_h(s_1,d_1) * \ldots * P_h(s_n,d_n)$$
$$= P_h(s,d)$$

□

Now that we have defined how the probability distribution of all nodes and in particular the nodes under the top-level action look like we can derive the decision strategy of the approach.

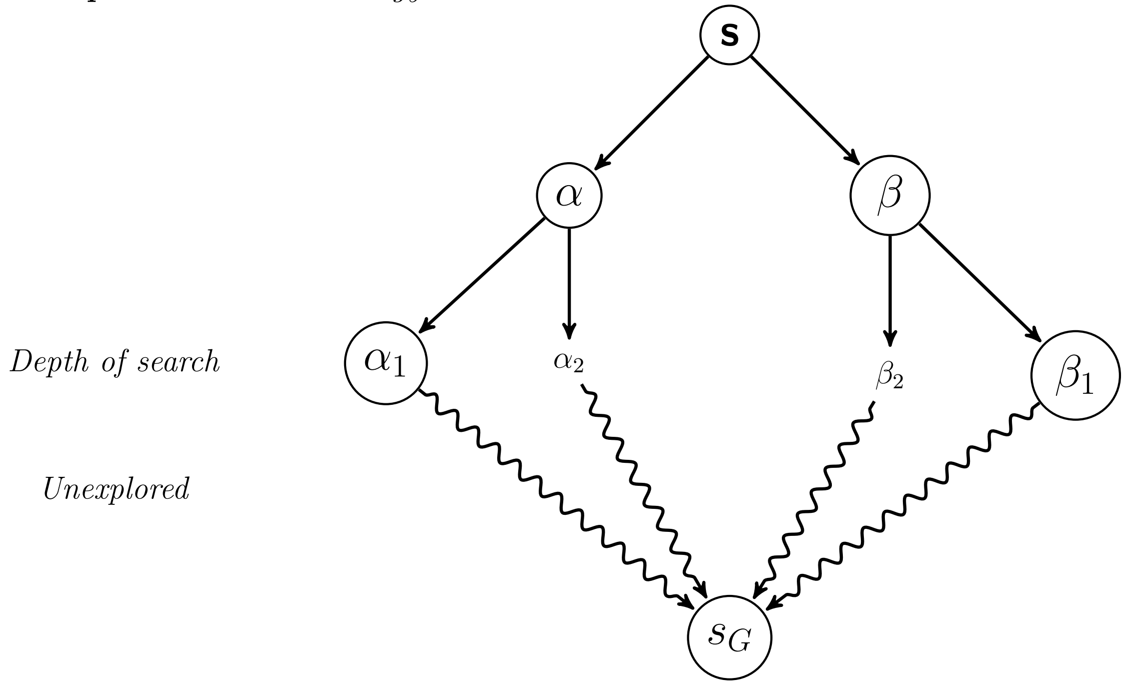
Definition 4. Decision Strategy

Given a node s with its child nodes and their corresponding probability distribution, the decision strategy is to go towards the node with the lowest expected value.

This definition makes sure that all possible outcomes of the shortest path are taken into account. The expected value is the usual expected value that is known from stochastic. It can be calculated as E(s) := $\sum^{p \in P_h(s,d)} p \cdot d$.

The following two example show how the decision strategy would decide in two different scenarios. Example 3 is a simple example, where it is shown that the expected value of the nodes does not correspond to the lowest possible value. Even though it could be better to go towards β the risk to go there is by far to high, since it is very much possible that the true distance from β to the goal is much longer than at α. Since this would be very bad in a setting where one has to choose a way and can not search further the decision strategy would choose to go towards α even though the potential shorted path would be through β.
Example 4 shows that it is not enough to just consider the expected value of the leaves nodes. The expected value of all leave nodes beneath α is lower than the expected value under all leaves nodes between β. However the expected value in α after doing the Cserna-Update is higher than the expected value in β. Thus the decision strategy would decide to go towards β. This behavior can be explained by the fact that β has more potential to become better when further search is done. Since in different nodes beneath β there are lower values than there are beneath α and the probability that all of them turn out bad is very low since they are all independent from each other (Assumption 3) we can expect that one of the paths beneath β is better than the paths beneath α.

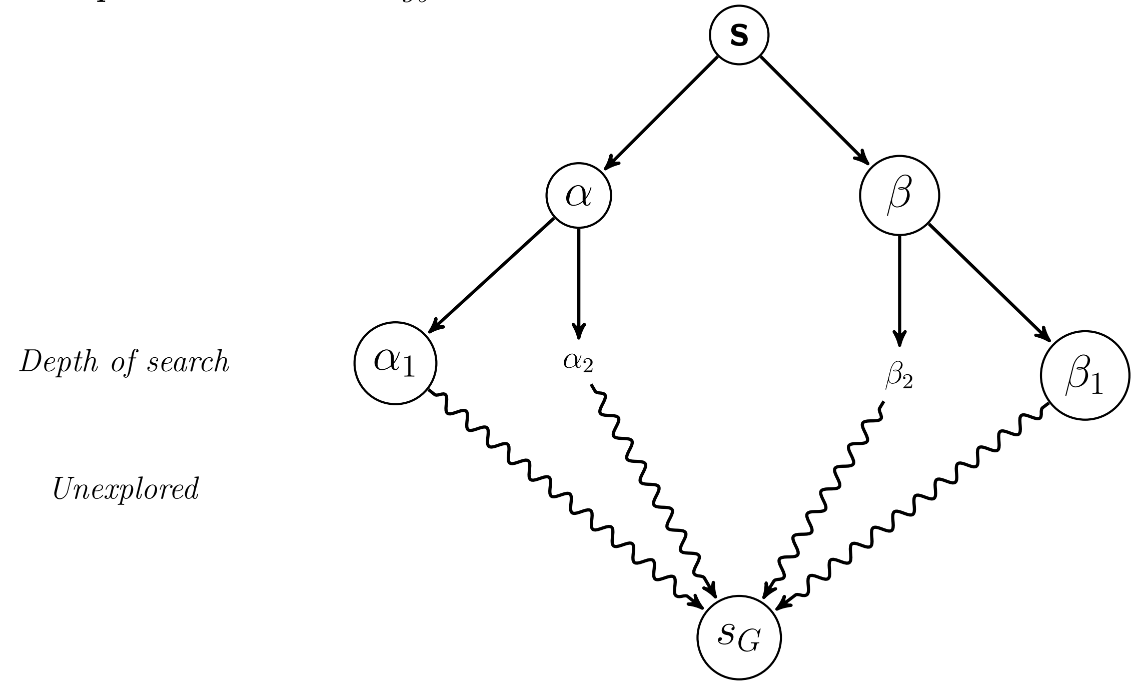
Example 3. *Decision Strategy*



If $P(\alpha_1) = P(\alpha_2) = \{(10, 1)\}$ *and*
$P(\beta_1)$=\{(8,0.5),(1000,0.5)\} and $P(\beta_2)$=\{(1000,0.5)\}
and all g = 0 in the depth of search
Since there exists so much uncertainty, whether the best possible value of β_1 can truly be acquired, and in case it does not, all paths given by β are much worse than the paths given by α, the decision strategy would choose to go towards α, since this decision is much safer and the expected cost of the path beneath α is lower than beneath β.

Calculation:
Expected Value:
$P(\alpha) = \{(10, 1)\}$
$P(\beta) = \{(6, 1 * 0.5), (1000, 1 * 0.5)\} = \{(6, 0.5), (1000, 0.5)\}$
*Thus the expected value of s_α is 10*1 = 10 and the expected value of s_β is 6*0.5+1000*0.5=503.*
Thus the decision strategy decides to go towards s_α

Example 4. *Decision Strategy:*



When $\hat{f}(\alpha_1) = \hat{f}(\alpha_2) = 20$ *with* $P(\alpha_1) = P(\alpha_2) = \{(16, 0.5), (24, 0, 5)\}$ *and*
$\hat{f}(\beta_1) = 19$*, but* $\hat{f}(\beta_2) = 24$*, while P(*β_1*)=*$\{$*(19,1)*$\}$ *and P(*β_2*)=*$\{$*(24,1)*$\}$
and all g = 0 in the depth of search
Since the $\hat{f}$ *value of* β *is just the better value of* β_1*,* β_2 *and thus 19, while the* $\hat{f}$ *value of* α *is better than both* α_1 *and* α_2*, since there are two nodes where the value 16 could be acquired, the decision strategy would choose to go towards* α*, since the expected value of* α *is lower than the expected value of* β*.*
Calculation:
Expected Value:
$P(\alpha) = \{(16, 0.5 * 0.5 * 3), (20, 0.5 * 0.5)\} = \{(16, 0.75), (20, 0.25)\}$
$P(\beta) = \{(19, 1)\}$
Thus the expected value of α *is 16*0.75+24*0.25 = 18 and the expected value of* β *is 19.*
Thus the decision strategy decides to go towards s_α

1.3 Exploration Strategy

Now that we have defined what we have to do when we have a certain knowledge given, now we have to define what the best way is to acquire knowledge. In particular this means we want to know which nodes should be expanded when the algorithm has to make a decision and what strategy of node expansions leads to a good approximation of these nodes. In order to reach this goal the first thing we have to do is to decide what function we want to optimize. One way one could do this is to maximize the difference between the expected value of the top-level action and all other actions. However we choose a different approach.

Definition 5. Confidence

The Confidence, that a selected top level action is optimal, is 1 minus the probability that any of the other possible actions is better. In other words given a state s, with successor states α, β, the Confidence that α is better than β is given by

$C(s_\alpha, s_\beta) = 1 - \sum_{d_\alpha, d_\beta \ s.t \ d_\alpha > d_\beta} P(s_\alpha, d_\alpha) * P(s_\beta, d_\beta)$

This Confidence thus describes the probability, that a top level action is at least as good as any other action. One should note that the node with the highest confidence is not necessarily the node with the lowest expected value. This can be be shown with a simple example of two nodes. One of the nodes has the probability distribution (10,1) while the other one has the probability distribution (8,0.4),(11,0.6). In this example the first bode has a higher confidence even though the second node has the lower expected value.
In the case that there are more than two nodes from which the confidence has to be calculated, the values should be added to the formula such that all d values are smaller than the d value it is compared with. the probability then gets multiplied to the result. in the case of three nodes α, β, γ this would take the following form:
$C(s_\alpha, s_\beta, s_\gamma) = 1 - \sum_{d_\alpha, d_\beta \ \gamma s.t \ d_\alpha > d_\beta, d_\alpha > d_\gamma} P(s_\alpha, d_\alpha) * P(s_\beta, d_\beta) * P(s_\gamma, d_\gamma)$

Example 5. *Calculating confidence with 3 nodes*
Given the following probability distributions
$P(s_\alpha, d) = (10, 0.5), (20, 0.5)$,
$P(s_\beta, d) = (14, 0.5), (16, 0.5)$,
$P(s_\gamma, d) = (12, 0.5), (18, 0.5)$

The confidence of s_α *is calculated as follows:*
$C(s_\alpha, s_\beta, s_\gamma) = 1 - P(s_\alpha, 10) * P(s_\beta, 14) * P(s_\gamma, 12) - P(s_\alpha, 10) * P(s_\beta, 16) * P(s_\gamma, 12) - P(s_\alpha, 10) * P(s_\beta, 14) * P(s_\gamma, 18) - P(s_\alpha, 10) * P(s_\beta, 16) * P(s_\gamma, 18) = 0.5$

The confidence of s_β *is calculated as follows:*
$C(s_\beta, s_\alpha, s_\gamma) = 1 - P(s_\alpha, 20) * P(s_\beta, 14) * P(s_\gamma, 18) - P(s_\alpha, 20) * P(s_\beta, 16) * P(s_\gamma, 18) = 0.25$

The confidence of s_γ *is calculated as follows:*
$C(s_\gamma, s_\alpha, s_\beta) = 1 - P(s_\alpha.20) * P(s_\beta, 14) * P(s_\gamma, 12) - P(s_\alpha, 20) * P(s_\beta, 16) * P(s_\gamma, 12) = 0.25$

What should be further noted is that when one adds the confidences of all nodes together, one does not necessarily get 1. In fact this happens only in the special case where no Probabilities greater 0 of two different nodes correspond to the same value. This can be easily seen in the formula by the fact that the value that are the same do not appear in the formula and thus do not get subtracted from 1.
This definition of the confidence can now be used to define the goal we want to achieve with the exploration strategy.

Definition 6. Optimal search tree

The optimal search tree of size n, is given by the search tree with n expanded nodes, where the confidence that the Decision strategy takes the optimal top level action is maximized.

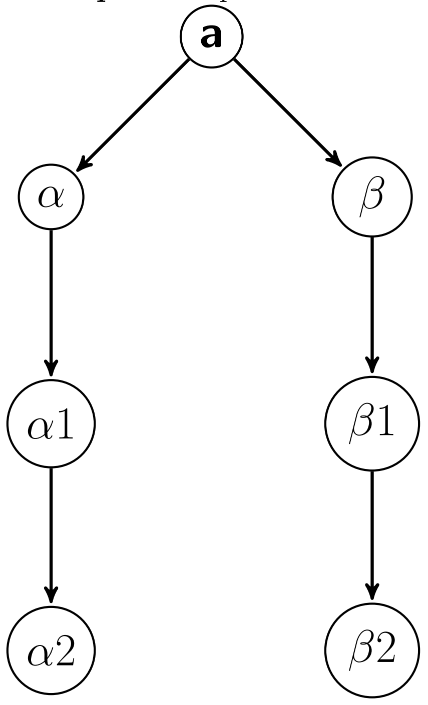
This definition is about the goal we try to achieve. Optimal means in this context that this is the best possible thing we can achieve. However even this definition does not guarantee that we make the best possible move. It just means that we choose our search tree so that the likelihood that we choose the best possible top-level action is maximized. Since we do not care about later steps when we make the decision which action we choose to maximize the Confidence that the first action is optimal is the best possible thing we can do. What should be noted is that it is not possible to have an strategy that always finds the optimal search tree that does not compare all possible search tree of size n. The reason for this is depicted in Example 6.
The goal of this definition is to be able to evaluate the exploration strategy. In case that the exploration strategy is able to find the optimal search tree often or the differences are small the exploration strategy is good. The bigger the differences are the more the search tree found by the exploration strategy differs from the optimality. Now that we have defined what is the goal we can define our exploration strategy, which goal it is to approximate the goal as best as possible.

Definition 7. Exploration Strategy

The Exploration Strategy should expand nodes with the goal that the optimal search tree is reached. For this purpose, the exploration strategy is to expand the nodes where the change in the confidence is maximized.

This exploration strategy makes sure that we always make the expansion that improves the search tree maximally with this one expansion. However as was said before this does not necessarily lead to the optimal search tree. since when expanding nodes one has now way of knowing what one will find in the future one can only do the best thing that can be seen at the moment of the expansion. Furthermore this exploration has one nice property in regards to the optimal search tree. This property is that if the tree found by the strategy of size n-1 is a subtree of the optimal search tree then this strategy will lead to the optimal search tree. This property can easily be verified by the fact that would an other node be expanded, then the increase in the confidence would be larger and thus the resulting confidence would be larger. However this is a contradiction to the fact that the confidence of the optimal search tree is maximized.
This property leads also to a case in which we can be sure to find the optimal search tree. This is when all optimal search trees of size smaller n are subtrees of the optimal search tree.

Example 6. *Optimal Search tree:*



If P(α)={(2,0.5),(10,0.25)},
P(β)={(2,0.5),(6,0.5)}
If P(α_1)= {(2,0.75),(10,0.25)} P(β_1)={(2,0.5),(6,0.5)}
P(α_2)={(2,0.75),(10,0.25)}
P(β_2)={(2,1)}

Since α and β are top-level actions they are already expanded.
The optimal search tree with n=4 has α_1 expanded, since then the confidence, that α is better is 0.75.
However the optimal search tree of size n=5 has β_1 and β_2 expanded, since then the confidence, that β is the best possible action is 1.
Thus the optimal search tree of size 4 is no subtree of the optimal search tree of size 5.

One further thing that is worth considering is whether there exist cases where we can be sure that expanding a node does not lead to a better confidence. One such case is when the best possible value in the given node is higher then the worst possible value in any other node. In other words We can be sure that expanding node n does not lead to an improvement of the confidence if:
min(P(n))>max(P(any node $\neq$ n))
In order to show this there are two cases that have to be considered.
Case 1:
Both nodes are under the same top-level action.
In this case the Cserna Update will ignore all values that are worse than the worst value of the better node. Thus the value will not reach the Probability distribution in the top-level action and thus the value does not appear when calculating the confidence.
Case 2:

Both nodes are under different top-level actions.
In this case the value reaches the Probability Distribution of the top-level action(and thus also influences the expected value), however since all values in the probability distribution of the other node are lower, the action does not influence the confidence.

A closer look at the second case also reveals an interesting behavior. While the confidence can not change by expanding the node, the expected value of the top-level action might change. Since a change in the expected value can lead to a change in the chosen path, it might happen that the path chosen by the decision strategy changes even though none of the confidences have changed. However since the exploration stratgey only cares about the confidences and not the expected values, the exploration strategy does not expand the node. Example 7 shows this behavior based of an example.

Example 7. *Node expansion leads to no change in confidence but in expected value*
$P(\alpha) = \{(10, 1)\}$
$P(\beta_1) = \{(8, 0.5), (15, 0.5)\}$
$P(\beta_2) = \{(11, 0.5), (15, 0.5)\}$
$=> P(\beta) = \{(8, 0.5), (11, 0.25), (15, 0.25)\}$
$=> E(\alpha) = 10$ *and* $E(\beta) = 8 * 0.5 + 11 * 0.25 + 15 * 0.25 = 10.5$
The Confidence of α is 0.5 and the Confidence of β is also 0.5

Suppose β_2 is expanded to β_3, with $P(\beta_3)$=$\{(11,1)\}$:
$=> P(\beta) = \{(8, 0.5), (11, 0.5)\}$
$=> E(\alpha) = 10$ *and* $E(\beta) = 8 * 0.5 + 11 * 0.5 = 9.5$
The Confidence of α is still 0.5 and the Confidence of β is also still 0.5