Available online at www.sciencedirect.com

ScienceDirect

International Journal of Approximate Reasoning
49 (2008) 316–330

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

www.elsevier.com/locate/ijar

Approximate reasoning in the algebra of bounded rational agents

Eugene Eberbach [1]

Department of Engineering and Science, Rensselaer Polytechnic Institute, 275 Windsor Street, Hartford, CT 06120, USA

Received 6 March 2006; received in revised form 4 September 2006; accepted 15 September 2006
Available online 22 October 2007

Abstract

In this paper, we generalize the utility theory to allow to use various performance measures, including utilities, costs and fitness, and probability theory we extend to uncertainty theory, including probabilities, fuzzy sets and rough sets. The decision theory is defined typically as the combination of utility theory and probability theory. We generalize the decision theory as the performance measure theory and uncertainty theory. Bounded rational agents look for approximate optimal decisions under bounded resources and uncertainty. The \$-calculus process algebra for problem solving applies the cost performance measures to converge to optimal solutions with minimal problem solving costs, and allows to incorporate probabilities, fuzzy sets and rough sets to deal with uncertainty and incompleteness. The approach is illustrated to find the optimal solutions with or without uncertainty. The same approach can be used to find solutions of the totally optimization problem, representing the tradeoff between the best quality and least costly solutions.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Approximate reasoning; Uncertainty; Bounded rational agents; Problem solving; Decision theory; Process algebra

1. Introduction

The decision theory is defined typically as the combination of utility theory and probability theory. In this paper we generalize the decision theory as the performance measure theory and uncertainty theory.

AI typically deals with dynamic, incomplete and uncertain domains where conventional algorithms do not perform well because of intractability or even undecidability. If so, the need for the new computational theory serving better new real-world applications and not hampered by computational explosion is obvious. Simply, in the solution of computational problems, the complexity of the reasoning process/search should be taken into account.

Von Neumann and Morgenstern [26] in 1944 gave the foundations of the decision theory using utilities and probabilities. Fifty years later Russell and Norvig [24] in the most popular AI textbook argued that the decision theory = utility theory + probability theory. However, their book describes much more than

[1] Research initially supported by ONR under grant N00014-06-1-0354 and initiated while at University of Massachusetts Dartmouth, eeberbach@umassd.edu. The paper presents an extended and modified version from the RSFDGRC'05 conference paper.
E-mail address: eberbe@rpi.edu

the utility theory and probability theory. Over 50 years, computer science and AI, in particular, have been changed dramatically. In this paper, we argue that the old approach to decision theory has to be extended to incorporate new approaches to optimization and uncertainty.

We generalize the utility theory to allow to use various performance measures, including utilities, costs and fitness, and probability theory we extend to uncertainty theory, including probabilities, fuzzy sets and rough sets.

Resource-based reasoning [20,24], called also anytime algorithms, trading off the quality of solutions for the amount of resources used, seems to be particularly well suited for the solution of hard computational problems in real time and under uncertainty. Additionally, new superTuring models of computation [11,13] trying to provide non-algorithmic solutions to the TM undecidable problems, can and should be useful for solutions of real-world problems. On the other hand, process algebras [21] are currently the most mature approach to concurrent and distributed systems, and seem to be the appropriate way to formalize multiagent systems.

The \$-calculus, presented in this paper, belongs to superTuring models of computation and provides support to handle intractability and undecidability in problem solving. Turing Machines and algorithms have been dominating computer science since 1960s. However, in 1990s it turned out that many areas require stronger models, and that Turing Machines and recursive algorithms do not represent a definitive limit of problem solving. In particular, the dynamic interaction of clients and servers on Internet, robot sensing and acting, and an infinite adaptation from evolutionary computation cannot be properly described using TMs and recursive algorithms. They require a violation of some or all requirements of recursive algorithms: the finiteness and/or well defined meaning of each step. The TM model is to weak to describe properly Internet, evolution, or robotics, because it is a closed model, which requires that all inputs are given in advance, and the TM is allowed to use an unbounded but only a finite amount of resources at a time (e.g., time or memory). By *superTuring computation* we mean any computation that cannot be carried out by a Turing Machines as well as any (algorithmic) computation carried out by a Turing Machine. Algorithms and computers that are more powerful than Turing Machines are called super-recursive algorithms and superTuring computers, respectively. Examples of superTruing models of computation include Turing's c-machines, o-machines, u-machines, von Neumann's cellular automata, interaction machines, neural networks, site and internet machines, inductive turing machines, persistent turing machines, evolutionary turing machines, π-calculus, \$-calculus. For more details, the reader is referred to [11,13,25,3]. In this paper, we present the \$-calculus to represent uncertainty and approximate reasoning, and we do not discuss its expressiveness. However, the precise optimal solutions found as the limit in infinite number of steps, and discussed in this paper, belong to super-recursive algorithms [3]. The expressiveness of \$-calculus, i.e., its super-recursive features are discussed in more details in [8,18].

Technically, the \$-calculus is a process algebra derived from Milner's π-calculus [21] extended by von Neumann/Morgenstern's costs/utilities [26] and a very general search method, called the $k\Omega$-optimization [16]. This novel search method allows to simulate many other search algorithms (of course, not all), including A^*, minimax, expectiminimax, hill climbing, dynamic programming, evolutionary algorithms, neural networks. The search tree can be infinite – this in the limit allows to solve non-recursively some undecidable problems (for instance, the halting problem of the universal turing machines, or to approximate a nonexisting universal search algorithm).

For solutions of intractable problems the total optimization is utilized to provide an automatic way to deal with intractability by optimizing together the quality of solutions and search costs.

This paper is organized as follows. In Section 2, the outline of problem solving performance by optimization under bounded resources is presented. In particular, the notions of completeness, optimization, search optimization, and total optimization are discussed. Section 3 contains necessary details of the \$-calculus, needed for understanding the following sections. The syntax and semantics based on the $k\Omega$-optimization are presented. The standard cost performance measure has been defined. Section 4 summarizes basic results allowing to find solutions at all (completeness), to find the best quality solutions (optimality), to find any solutions with minimal amount of time or memory (search optimality), and, finally, to find the best quality solutions with minimal amount of resources used (total optimality). Section 5 illustrates the \$-calculus approach to problem solving without uncertainty using as an example the A^* search algorithm simulated by the \$-calculus $k\Omega$-meta search. Next, the A^* is extended to AU^* (A^* with uncertainty) and solved by applying the \$-calculus $k\Omega$-meta search with standard cost function incorporating either probabilities, or fuzzy set or rough set membership functions. Section 6 contains conclusions and problems to be solved in the future.

2. Measuring problem solving performance: optimization under bounded resources

The performance of search algorithms (intelligence of an agent) can be evaluated in four ways (see e.g. [24]) capturing whether a solution has been found, its quality and the amount of resources used to find it.

Definition 1 (*On completeness, optimality, search optimality, and total optimality*). We say that the search algorithm is

– **Complete** if it guarantees reaching a terminal state/solution if there is one.
– **Optimal** if the solution is found with the optimal value of its objective function.
– **Search Optimal** if the solution is found with the minimal amount of resources used (e.g., the time and space complexity).
– **Totally Optimal** if the solution is found both with the optimal value of its objective function and with the minimal amount of resources used.

Definition 2 (*On problem solving as a multiobjective minimization problem*). Given an objective function $f: A \times X \to R$, where A is an algorithm space with its input domain X and codomain in the set of real numbers, R, problem solving can be considered as a multiobjective minimization problem to find $a^* \in A_F$ and $x^* \in X_F$, where $A_F \subseteq A$ are terminal states of the algorithm space A, and $X_F \subseteq X$ are terminal states of X such that

$$f(a^*, x^*) = \min\{f_1(f_2(a), f_3(x)), a \in A, x \in X\},$$

where f_3 is a problem-specific objective function, f_2 is a search algorithm objective function, and f_1 is an aggregating function combining f_2 and f_3.

Without losing generality, it is sufficient to consider only minimization problems. An objective function f_3 can be expanded to multiple objective functions if the problem considered has several objectives. The aggregating function f_1 can be arbitrary (e.g., additive, multiplicative, a linear weighted sum). The only requirement is that it captures properly the dependence between several objectives. In particular, if f_1 becomes an identity function, we obtain the Pareto optimality

$$f(a^*, x^*) = \min\{(f_2(a), f_3(x)), a \in A, x \in X\}.$$

Using Pareto optimality is simpler, however we lose an explicit dependence between several objectives (we keep a vector of objectives ignoring any priorities, on the other hand, we do not have problems combining objectives if they are measured in different "units", for example, an energy used and satisfaction of users). For fixed f_2 we consider an optimization problem – looking for minimum of f_3, and for fixed f_3 we look for minimum of search costs – search optimum of f_2.

Objective functions allow capturing convergence and the convergence rate of construction of solutions much better than symbolic goals. Obviously every symbolic goal/termination condition can be expressed as an objective function. For example, a very simple objective function can be the following: if the goal is satisfied the objective is set to 1, and if not to 0. Typically, much more complex objective functions are used to better express evolutions of solutions.

Let (A^*, X^*) denotes the set of totally optimal solutions. In particular X^* denotes the set of optimal solutions, and A^* the optimal search algorithms.

Let Y be a metric space, where for every pair of its elements x, y there is assigned the real number $D(x,y) \geqslant 0$, called *distance*, satisfying three conditions:

1. $D(x,x) = 0$,
2. $D(x,y) = D(y,x)$,
3. $D(x,y) + D(y,z) \geqslant D(x,z)$.

The distance function can be defined in different ways, e.g., as the Hamming distance, Euclidean distance, $D(x) = 0$ if x satisfies termination condition and $D(x) = 1$ otherwise. To keep it independent from representation, and to allow to compare different solving algorithms, we will fix the distance function to the absolute value of difference of the objective functions $D(x,y) = |f(x) - f(y)|$. We extend the definition of the distance from the pairs of points to the distance between a point and the set of points $D(x, Y) = \min\{|f(x) - f(y)|; y \in Y\}$

In problem solving, we will be interested in the distance to the set of optimal solutions Y^*, i.e., in the distance $D((a,x),(A^*,X^*))$, and in particular $D(x,X^*)$, $D(a,A^*)$, where $x \in X$ is the solution of the given problem instance, and $a \in A$ is the algorithm producing that solution.

Definition 3 (*On solution convergence*). For any given problem instance, its solution evolved in the discrete time $t = 0,1,2,\ldots$, will be said to be

– *convergent* to the total optimum iff there exists such τ that for every $t > \tau$ $D((a[t],x[t]),(A^*,X^*)) = 0$,
– *asymptotically convergent* to the total optimum iff for every ε, $\infty > \varepsilon > 0$, there exists such τ that for every $t > \tau$ $D((a[t],x[t]),(A^*,X^*)) < \varepsilon$,
– *convergent with an error* ε to the total optimum, where $\infty > \varepsilon > 0$ iff there exists such τ that for every $t > \tau$ $D((a[t],x[t]),(A^*,X^*)) \leqslant \varepsilon$,
– *divergent*, otherwise.

If solution is convergent and τ is fixed, then the convergence is recursive, otherwise it is superrecursive. Asymptotic convergence is superrecursive (the time is unbounded).

Search can involve single or multiple agents. For multiple agents search can be *cooperative*, *competitive*, or *random*. In cooperative search other agents help to find an optimum, in competitive search – they distract to reach an optimum, and in random search other agents do not care about helping or distracting to reach an optimum. Search algorithms can be *online*, where action execution and computation are interleaved, and *off-line*, where the complete solution is computed first and executed after without any perception.

3. The $-calculus algebra of bounded rational agents

The $-calculus is a mathematical model of processes capturing both the final outcome of problem solving as well as the interactive incremental way how the problems are solved. The $-calculus is a process algebra of bounded rational agents for interactive problem solving targeting intractable and undecidable problems. It has been introduced in the late of 1990s [9,11,13,25,15,16]. The $-calculus (pronounced COST calculus) is a formalization of resource-bounded computation (also called anytime algorithms), proposed by Dean, Horvitz, Zilberstein and Russell in the late 1980s and early 1990s [20,24]. Anytime algorithms are guaranteed to produce better results if more resources (e.g., time, memory) become available. The standard representative of process algebras, the π-calculus [21] is believed to be the most mature approach for concurrent systems.

The $-calculus rests upon the primitive notion of *cost* in a similar way as the π-calculus was built around a central concept of *interaction*. Cost and interaction concepts are interrelated in the sense that cost captures the quality of an agent interaction with its environment. The unique feature of the $-calculus is that it provides a support for problem solving by incrementally searching for solutions and using cost to direct its search. The basic $-calculus search method used for problem solving is called $k\Omega$-optimization. The $k\Omega$-optimization represents this "impossible" to construct, but "possible to approximate indefinitely" universal algorithm. It is a very general search method, allowing the simulation of many other search algorithms, including A^*, minimax, dynamic programming, tabu search, or evolutionary algorithms. Each agent has its own Ω search space and its own limited horizon of deliberation with depth k and width b. Agents can cooperate by selecting actions with minimal costs, can compete if some of them minimize and some maximize costs, and be impartial (irrational or probabilistic) if they do not attempt optimize (evolve, learn) from the point of view of the observer. It can be understood as another step in the never ending dream of universal problem solving methods recurring throughout all computer science history. Because the $-calculus, similar like Turing Machine, is a generic tool for problem solving, its applications are enormous. The $-calculus is applicable to robotics [7,12], software agents, neural nets, and evolutionary computation [10]. In particular, it has been applied for the design of special-purpose CCL language to control the group of undersea autonomous vehicles [5,12,6] and general-purpose CO$T language for problem solving [14]. It has been used to simulate by one algorithm – the $k\Omega$-optimization both single and multiple, global and local sequence alignment algorithms [1,2], to model polymorphic viruses [17]. Potentially $-calculus could be used for design of cost languages [14], cellular evolvable cost-driven hardware, DNA-based computing and bioinformatics [1], electronic commerce, data mining, machine vision, and quantum computing [16]. The $-calculus leads to a new programming paradigm *cost languages* [14] and a new class of computer architectures *cost-driven computers*.

3.1. The $-calculus syntax

In $-calculus everything is a cost expression: agents, environment, communication, interaction links, inference engines, modified structures, data, code, and meta-code. $-Expressions can be simple or composite. Simple $-expressions α are considered to be executed in one atomic indivisible step. Composite $-expressions P consist of distinguished components (simple or composite ones) and can be interrupted.

Definition 4 (*The* $-*calculus*). The set $\mathscr{P}$ of $-calculus process expressions consists of simple $-expressions α and composite $-expressions P, and is defined by the following syntax:

α	$::=$	$(\$_{i\in I}\ P_i)$	cost
	$\vert$	$(\rightarrow_{i\in I}\ c\ P_i)$	send P_i with evaluation through channel c
	$\vert$	$(\leftarrow_{i\in I}\ c\ X_i)$	receive X_i from channel c
	$\vert$	$('_{i\in I}\ P_i)$	suppress evaluation of P_i
	$\vert$	$(a_{i\in I}\ P_i)$	defined call of simple $-expr. a with parameters P_i
	$\vert$	$(\bar{a}_{i\in I}\ P_i)$	negation of defined call of simple $-expression a
P	$::=$	$(\circ_{i\in I}\ \alpha\ P_i)$	sequential composition
	$\vert$	$(\Vert_{i\in I}\ P_i)$	parallel composition
	$\vert$	$(\Cup_{i\in I}\ P_i)$	cost choice
	$\vert$	$(\uplus_{i\in I}\ P_i)$	adversary choice
	$\vert$	$(\sqcup_{i\in I}\ P_i)$	general choice
	$\vert$	$(f_{i\in I}\ P_i)$	defined process call f with parameters P_i, and its associated definition $(:=(f_{i\in I}\ X_i)R)$ with body R

The indexing set I is a possibly countably infinite. In the case when I is empty, we write empty parallel composition, general, cost and adversary choices as $\perp$ (blocking), and empty sequential composition (I empty and $\alpha = \varepsilon$) as ε (invisible transparent action, which is used to mask, make invisible parts of $-expressions). Adaptation (evolution/upgrade) is an essential part of $-calculus, and all $-calculus operators are infinite (an indexing set I is unbounded). The $-calculus agents interact through send-receive pair as the essential primitives of the model.

Sequential composition is used when $-expressions are evaluated in a textual order. Parallel composition is used when expressions run in parallel and it picks a subset of non-blocked elements at random. Cost choice is used to select the cheapest alternative according to a cost metric. Adversary choice is used to select the most expensive alternative according to a cost metric. General choice picks one non-blocked element at random. General choice is different from cost and adversary choices. It uses guards satisfiability. Cost and adversary choices are based on cost functions. Call and definition encapsulate expressions in a more complex form (like procedure or function definitions in programming languages). In particular, they specify recursive or iterative repetition of $-expressions.

Simple cost expressions execute in one atomic step. Cost functions are used for optimization and adaptation. The user is free to define his/her own cost metrics. Send and receive perform handshaking message-passing communication, and inferencing. The suppression operator suppresses evaluation of the underlying $-expressions. Additionally, a user is free to define her/his own simple $-expressions, which may or may not be negated.

3.2. The $-calculus semantics: the $k\Omega$-search

In this section we define the operational semantics of the $-calculus using the $k\Omega$-search that captures the dynamic nature and incomplete knowledge associated with the construction of the problem solving tree.

The basic \$-calculus problem solving method, the $k\Omega$-optimization, is a very general search method providing meta-control, and allowing to simulate many other search algorithms, including A^*, minimax, dynamic programming, tabu search, or evolutionary algorithms [24]. The problem solving works iteratively: through select, examine and execute phases. In the select phase the tree of possible solutions is generated up to k steps ahead, and agent identifies its alphabet of interest for optimization Ω. This means that the tree of solutions may be incomplete in width and depth (to deal with complexity). However, incomplete (missing) parts of the tree are modeled by silent \$-expressions ε, and their cost estimated (i.e., not all information is lost). The above means that $k\Omega$-optimization may be if some conditions are satisfied to be complete and optimal. In the examine phase the trees of possible solutions are pruned minimizing cost of solutions, and in the execute phase up to n instructions are executed. Moreover, because the \$ operator may capture not only the cost of solutions, but the cost of resources used to find a solution, we obtain a powerful tool to avoid methods that are too costly, i.e., the \$-calculus directly minimizes search cost. This basic feature, inherited from anytime algorithms, is needed to tackle directly hard optimization problems, and allows to solve total optimization problems (the best quality solutions with minimal search costs). The variable k refers to the limited horizon for optimization, necessary due to the unpredictable dynamic nature of the environment. The variable Ω refers to a reduced alphabet of information. No agent ever has reliable information about all factors that influence all agents behavior. To compensate for this, we mask factors where information is not available from consideration; reducing the alphabet of variables used by the \$-function. By using the $k\Omega$-optimization to find the strategy with the lowest \$-function, meta-system finds a satisfying solution, and sometimes the optimal one. This avoids wasting time trying to optimize behavior beyond the foreseeable future. It also limits consideration to those issues where relevant information is available. Thus the $k\Omega$-optimization provides a flexible approach to local and/or global optimization in time or space. Technically this is done by replacing parts of \$-expressions with invisible \$-expressions ε, which remove part of the world from consideration (however, they are not ignored entirely – the cost of invisible actions is estimated).

Definition 5 (*The $k\Omega$-Optimization meta-search procedure*). The $k\Omega$-optimization meta-search procedure $k\Omega_i[t]$ for the ith agent, $i = 0, 1, 2, \ldots$, from an enumerable *universe of agent population* and working in time generations $t = 0, 1, 2, \ldots$ is a complex \$-expression (meta-procedure) consisting of simple \$-expressions $init_i[t], sel_i[t]$, $exam_i[t], goal_i[t]$, $\$_i[t]$, complex \$-expression $loop_i[t]$ and $exec_i[t]$, and constructing solutions, its input $x_i[t]$, from predefined and user defined simple and complex \$-expressions. For simplicity, we will skip time and agent indices in most cases if it does not cause confusion, and we will write *init*, *loop*, *sel*, *exam*, $goal_i$ and $\$_i$. Each ith agent performs the following $k\Omega$-search procedure $k\Omega_i[t]$ in the time generations $t = 0, 1, 2, \ldots$

```
(:=($k\Omega_i[t]$ $x_i[t]$)($\circ$(init ($k\Omega_i[0]$ $x_i[0]$))   // initialize $k\Omega_i[0]$ and $x_i[0]$
      (loop $x_i[t+1]$))                      // basic cycle: select, examine, execute
)
```

where *loop* meta-\$-expression takes the form of the select-examine-execute cycle performing the $k\Omega$-optimization until the goal is satisfied. At that point, the agent re-initializes and works on a new goal in the style of the never ending reactive program:

```
(:=(loop $x_i[t]$)                   // loop recursive definition
   ($\sqcup$($\circ$($\overline{goal_i[t]}$($k\Omega_i[t] x_i[t]$))      // goal not satisfied, default goal min($\$_i$($k\Omega_i[t]$ $x_i[t]$))
        (sel $x_i[t]$)               // select: build problem solution tree k step deep, b wide
        (exam $x_i[t]$)              // examine: prune problem solution tree in cost and in
                                     adversary choices
        (exec($k\Omega_i[t]$ $x_i[t]$))      // execute: run optimal $x_i$ $n$ steps and update $k\Omega_i$ param.
        (loop $x_i[t+1]$))           return back to loop
      ($\circ$ ($goal_i[t]$($k\Omega_i[t]$ $x_i[t]$))     goal satisfied – re-initialize search
        ($k\Omega_i[t]$ $x_i[t]$)))
)
```

More details on the $k\Omega$-search, including inference rules of the labeled transition system, observation and strong bisimulations and congruences, can be found in [16].

3.3. Probabilistic, fuzzy sets and rough sets performance measure

The domain of the cost function is a problem-solving derivation tree constructed by the $k\Omega$-optimization meta-procedure. The derivation tree consists of nodes/states $\mathscr{S}$ and edges/actions $\mathscr{E}$. Both $k\Omega_i[t]$ and $x_i[t]$ \$-expressions form own trees, where $k\Omega_i[t]$ tree is responsible for generation, pruning and evaluation of $x_i[t]$ tree representing a problem solution. The $x_i[t]$ tree is problem-specific. On the other hand, the $k\Omega_i[t]$ tree has a fixed form $init_i$ followed by two branches $\overline{goal_i}$, sel_i, $exam_i$, $exec_i$,$\overline{goal_i}$, sel_i, $exam_i$, $exec_i, \ldots$, and $goal_i$, $k\Omega_i[t]$. To avoid the complexity to analyze and synchronize two trees for total optimization, both trees can be compressed/collapsed into a single tree, where states/nodes represent a Cartesian product of states of $x_i[t]$ and $k\Omega_i[t]$, and edges/actions are labeled by Cartesian product of simple \$-expressions from $x[t]$ and corresponding \$-expressions from $k\Omega[t]$. We will denote by $h : x[t] \rightarrow k\Omega[t]$ an isomorphism mapping $x[t]$ to corresponding $k\Omega[t]$. Each agent will have each own such single "doubled" tree, representing two trees, i.e., the population of agents will be represented by a vector of trees (a forest). In such a way, a problem-solving tree will capture both solutions and the search process. For optimization or search optimization, one component in the doubled tree (either $x[t]$ or $k\Omega[t]$) can be omitted.

The cost function $\$_3$ measures the quality of solutions (costs of $x_i[t]$), and sometimes only the costs of subtree (for example cost of leaves representing the final complete solutions if $k\Omega[t]$ works in the style of evolutionary computation with $gp = 1$). The cost function $\$_2$ measures the costs of search (costs of $k\Omega_i[t]$), and the $\$_1$ aggregating function combines costs of solutions and search.

Let's define the cost of the problem-solving tree $\mathscr{T}$ recursively as a cost of the root node, cost of transitive actions leading to new states, and cost of corresponding subtrees, with new states being their roots. This will combine costs of the search and the solution quality as $\$: \mathscr{T} \rightarrow \mathscr{R}^\infty$, where $\mathscr{R}^\infty = \mathscr{R} \cup \{\infty\}$, i.e.,

$$\$(k\Omega_i[t], x_i[t]) = \$_1(\$_2(h(x_i[t])), \$_3(x_i[t])) = \$_1(\$_2(k\Omega_i[t]), \$_3(x_i[t])),$$

where $\$_1$ is an aggregating cost function, $\$_2$ is a search cost function, and $\$_3$ is the problem-specific cost function.

In this paper, both $\$_2$ and $\$_3$ will take the same uniform form of the standard cost function defined below, and $\$_1$ will take a form of addition, i.e., $\$(k\Omega_i[t], x_i[t]) = \$_2(k\Omega_i[t]) + \$_3(x_i[t])$. Without compromising a generality, it is sufficient to define costs of \$-expressions $\mathscr{P}$ for either $\$_2$ or $\$_3$, because cost of solutions and search will be simply a sum of both $\$_2$ and $\$_3$. Technically, $\$_j$ is defined on the problem-solving tree, consisting of nodes and edges expressed by \$-expressions, and representing both a solution and meta-search procedure, as the function mapping the tree to a real number: $\$_j : \mathscr{P} \rightarrow \mathscr{R}^\infty,\ j = 2, 3$.

Let $v : \mathscr{A}^\varepsilon \rightarrow \mathscr{R}^\infty$ be costs of simple cost expressions, including a silent expression. They are context dependent, i.e., they depend on states. In particular, cost of ε may depend which cost expression is made invisible by ε. Technically, \$ is defined on the problem-solving tree, consisting of nodes and edges expressed by \$-expressions, as the function mapping the tree to a real number: $\$_i : \mathscr{P} \rightarrow \mathscr{R}^\infty,\ i = 2, 3$. Thus it is sufficient to define costs of \$-expressions $\mathscr{P}$. Note that the value of the cost function (or its estimate) can change after each loop iteration (evaluation of a simple cost expression).

Definition 6 (*A standard cost function*). For every \$-expression P its cost $(\$_j P)$, $j = 1, 2, 3$ is defined as below:

1. $(\$_j \perp) = +\infty$;
2. $(\$_j\ \varepsilon) = \begin{cases} 0 & \text{for observation congruence,} \\ (v\ \varepsilon) & \text{for strong congruence;} \end{cases}$
3. $(\$_j\ \alpha) = c_\alpha + (v\ \alpha)$, where $c_\alpha = \begin{cases} 0 & \alpha \text{ does not block,} \\ +\infty & \alpha \text{ blocks,} \end{cases}$ $(\$_j\ \bar{\alpha}) = \frac{1}{c_\alpha} + (v\ \bar{\alpha})$;
4. $(\$_j\ (\sqcup_{i \in I} P_i)) = \begin{cases} \Sigma_{i \in I}\ (p_i * (\$_j P_i)) & \text{for probability-based cost function,} \\ (\$_j\ P_l), l = \text{argmax}_{i \in I}\ m_i & \text{for fuzzy set-based cost function,} \\ (\$_j\ P_l), l = \text{argmax}_{i \in I}\ \mu_i & \text{for rough set-based cost function,} \end{cases}$

   where p_i is the probability of choice of the ith branch, m_i is a fuzzy set membership function of choice of the ith branch, and μ_i is a rough set membership function of the ith branch choice
5. $(\$_j\ (\Cup_{i \in I} P_i)) = (\min_{i \in I}\ (\$_j\ P_i))$;
6. $(\$_j\ (\uplus_{i \in I} P_i)) = (\max_{i \in I}\ (\$_j\ P_i))$;

7. $(\$_j\ (\circ_{i\in I}\ \alpha\ P_i)) = (\$_j\ \alpha) + \Sigma_{i\in I}\ (\$_j\ P_i')$, where P_i' represents a possible change of P_i by receive or return value by α;

8. $$(\$_j\ \|_{i\in I}\ P_i) = \begin{cases} \Sigma_{J\subseteq I}\ p_J * ((\$_j\ \{\alpha_j\}_{j\in J}) + (\$_j\ (\|_{i\in I-J, j\in J}\ P_i\ P_j'))) \\ \text{for probability-based cost function,} \\ ((\$_j\ \{\alpha_j\}_{j\in L}) + (\$_j(\|_{i\in I-L, j\in L}\ P_i\ P_j'))) \\ L = \mathrm{argmax}_{J\subseteq I} m_J, \text{for fuzzy set-based cost function,} \\ ((\$_j\ \{\alpha_j\}_{j\in L}) + (\$_j(\|_{i\in I-L, j\in L}\ P_i\ P_j'))) \\ L = \mathrm{argmax}_{J\subseteq I} \mu_J, \text{for rough set-based cost function,} \end{cases}$$

   where p_J is the probability of choice of the Jth multiset, m_J is a fuzzy set membership function of choice of the Jth multiset, and μ_J is a rough set membership function of the Jth multiset choice;

9. $(\$_j\ (f_{i\in I}\ Q_i)) = (\$_j\ P\{Q_i/X_i\})$ where $(:=(f_{i\in I}\ X_i)P)$.

Cost choice calculates costs as the minimum of costs of its components. Adversary choice cost is defined as the cost of its most expensive component. General choice cost has been defined as the average component cost if to use probabilities to represent uncertainty, or the maximum if to use fuzzy sets [27] or rough sets [22]. Sequential composition cost adds costs of its components. Parallel composition cost selects a nonempty multiset that does not block. It has been defined as the average component cost. Alternatively, parallel composition could select a specific multiset to be executed, e.g., the maximum subset that does not block (for the maximum concurrency semantics), or the subset with the minimal costs (probably the most interesting alternative, on the other hand, increasing costs of the $k\Omega$-search). However, both these alternatives we will leave as viable choices for the user who can overwrite the cost of parallel composition definition if it is preferable. Cost of the recursive (user defined) function call is calculated as the cost of its body.

4. The \$-calculus support for intractability: optimization under bounded resources, completeness, optimality, search optimality and total optimality

It is important to determine whether the \$-calculus $k\Omega$-optimization will find a solution (will reach a terminal state – the goal of computation) if there is one, whether solutions are optimal, and what is the search cost associated with problem-solving, i.e., whether the \$-calculus search is complete, optimal, search optimal, and totally optimal. In particular, search optimality and total optimality is crucial for the solution of intractable problems, because it takes into account an amount of resources available for an agent.

Definition 7 (*On completeness of the* \$-*calculus search*). The \$-calculus search is complete if the $k\Omega$-optimization starting from its initial state $(k\Omega[0], x[0])$ guarantees to reach a state $(k\Omega[t], x[t])$ satisfying the goal condition pending there is one.

Formally, the completeness of search belongs to decision problems. Reaching an arbitrary terminal/goal state is equivalent to the halting problem of the Turing machine and is undecidable. Thus completeness of search guarantees that the problem of reaching the goal becomes decidable.

Definition 8 (*On elitist selection of the* \$-*calculus search*). The \$-calculus search will use an *elitist strategy* if states selected for expansion in the next loop iteration of the $k\Omega$-optimization will contain states with the most promising (i.e., minimal) costs.

Using elitism will allow to expand the most promising parts of the tree only.

Definition 9 (*On admissibility of the* \$-*calculus search*). The \$-calculus search will be *admissible* if the costs of silent \$-expressions are not overestimated.

The admissibility requirement will prohibit to stop prematurely search if a non-optimal goal is found that may look a more promising than the optimal goal. Note that elitist selection concept is typical for evolutionary algorithms, and admissibility for heuristic search, e.g., the A^* algorithm.

Definition 10 (*On optimality of the* \$-*calculus search*). The \$-calculus search of the ith agent is *optimal* if the $k\Omega$-optimization has its goal condition set to the optimum of the problem-specific cost function and $\$_{3i}(x_i[t])$ is convergent (or asymptotically convergent) to the set of optimal solutions X_i^*.

For on-line algorithms with $n > 0$, the optimal solution will be restricted to a specific n value of steps scheduled for execution in the execution phase. For off-line search algorithms with $n = 0$, the optimal solution will be the complete solution scheduled for execution in the execution phase.

Theorem 1 (On optimality of the \$-calculus search). *For a given $k\Omega$-optimization procedure $k\Omega_i[0]$ with an initial problem solution $x_i[0]$, if the* \$-*calculus search of the ith agent satisfies four conditions*:

1. *the goal condition is set to the optimum of the problem-specific cost function $\$_{3i}(x_i[t])$ with the optimum $\$^*_{3i}$*,
2. *search is complete*,
3. *elitist selection is used, and*
4. *search is admissible*,

then the $k\Omega$-optimization will find the optimum x_i^* of $\$_{3i}(x_i[t])$ in an unbounded number of iterations $t = 0,1,2,\ldots$, and that optimum will be maintained thereafter.

Proof. By completeness, the $k\Omega$-search reaches (perhaps in an infinite number of loop iterations) a goal state that is equivalent to the optimal state. By elitism the optimum will be maintained (i.e., not lost), because the rate of convergence is guranteed to be greater than or equal to zero. The rate of convergence cannot be permanently equal to 0, because if other more promising nodes exist, the admissibility will guarantee that they will have to be expanded. By admissibility the premature stopping in a local optimum will be prevented, because an optimal state will be always looking as a more promising. Always the most promising node (the cheapest one, according to the $\$_{3i}$ metric) will be in the group of nodes selected for expansion in a new iteration, because the $k\Omega$-optimization expands all nodes in order of increasing $\$_{3i}$ values, thus it must eventually expand the optimal (reachable) goal state. Both conditions imply that the $k\Omega$-optimization will eventually converge (asymptotically converge), perhaps requiring an infinite number of generations, to the optimum x_i^* of $\$_{3i}$. □

Definition 11 (*On search optimality of the* \$-*calculus search*). The \$-calculus search of the ith agent is search optimal if the $k\Omega$-optimization has its goal condition set to the optimum of the search cost function together with problem-specific goal condition, and $\$_{2i}(k\Omega_i[t])$ is convergent (or asymptotically convergent) to the set of optimal solutions $k\Omega_i^*$.

Theorem 2 (On search optimality of the \$-calculus search). *For a given $k\Omega$-optimization procedure $k\Omega_i[0]$ with an initial problem solution $x_i[0]$, if the* \$-*calculus search of the ith agent satisfies four conditions*:

1. *the goal condition requires additionally to reach the optimum of the search algorithm cost function $\$_{2i}(k\Omega_i[t])$ with the optimum $\$^*_{2i}$*,
2. *search is complete*,
3. *elitist selection is used, and*
4. *search is admissible*,

then the $k\Omega$-optimization will find the optimum $k\Omega_i^*$ of $\$_{2i}(k\Omega_i[t])$ in an unbounded number of iterations $t = 0,1,2,\ldots$, and that optimum will be maintained thereafter.

Proof. The proof is analogous to the proof of optimality. The only difference is that now costs of problem-specific solutions are ignored (costs of nodes are treated as 0), and costs of edges representing cost of the $k\Omega$-optimization are taken into account. Besides minimal costs of search, we still require that a specific problem is solved. □

A total optimality provides a direct and elegant method to deal with intractability of problem solving search. It will use a power of evolution to avoid expensive search methods. In other words, both the solutions and algorithms producing the solutions will be evolved (but for the price that perhaps the quality of solutions found would be worse compared to solutions where we ignore search costs, i.e., total optima in most cases are different than problem-specific optima).

Definition 12 (*On total optimality of the* \$-*calculus search*). The \$-calculus search of the ith agent is totally optimal if the $k\Omega$-optimization has its goal condition set to the optimum of the cost function $\$_i(k\Omega_i[t], x_i[t])$ and $\$_i(k\Omega_i[t], x_i[t])$ is convergent/asymptotically convergent to the set of optimal solutions $(k\Omega_i^*, X_i^*)$.

Theorem 3 (On total optimality of the \$-calculus search). *For a given kΩ-optimization procedure* $k\Omega_i[0]$ *with an initial problem solution* $x_i[0]$, *if the* \$-*calculus search of the ith agent satisfies four conditions*:

1. *the goal condition is set to the optimum of the search algorithm cost function* $\$_i(k\Omega_i[t], x_i[t])$ *with the optimum* $\$_i^*$,
2. *search is complete*,
3. *elitist selection is used, and*
4. *search is admissible*,

then the kΩ-optimization will find the optimum $(k\Omega_i^*, x_i^*)$ *of* $\$_i(k\Omega_i[t], x_i[t])$ *in an unbounded number of iterations* $t = 0, 1, 2, \ldots$, *and that optimum will be maintained thereafter.*

Proof. Analogous to theorem on optimality of the $k\Omega$-search. □

5. Approximate reasoning in environments with uncertainty

The real world problems by necessity (caused by complexity of the problem or environment, the lack or imprecise information) has to deal with uncertainty of sensors, actions, perceptions. There are various approaches to deal with uncertainty: probability theory, Zadeh's fuzzy sets, the Dempster–Shaffer theory of evidence, Pawlak's rough sets, Reiter's default logic, McCarthy's circumscription, or McDermott/Doyle's nonmonotonic logic [24]. All these methods are used separately, and it is very difficult to say which one is the best to express uncertainty. The most compiling result is de Finetti theorem [4,24] that claims that the probability methods always will outperform (will give higher payoffs) other methods that violate axioms of probability theory. However, that claim is based on the assumption that the real world is ruled by probabilities, and not for example by fuzzy sets or nonmonotonic logic.

For this reason, it would be desirable to have models that allow to compare and experiment with various approaches to uncertainty. The \$-calculus allows at this moment to use either probabilities, fuzzy sets or rough sets.

For illustration we will consider the robot trying to find the shortest path with or without uncertainty. The possible scenario to test the optimality/total optimality could be a robot navigating from the starting to the terminal point, where the trajectory of the robot is a subject to uncertainty (wheels are slippery, sensors measurements are imprecise). Thus we can interpret that the robot, instead of the desired point/state, may reach several points/states measured either with some probability, or fuzzy set, or rough set membership function. The robot tries to find the shortest path with or without uncertainty by solving the $k\Omega$-optimization problem, or to find a shortest path and, additionally, to minimize the time spent on the trajectory computation at the same time for the total optimization problems.

The A^* algorithm belongs to the best known examples of heuristic search (due to Hart et al., 1968, also called the best-path search [19]). A^* selects the path with the shortest length expanding the most promising node in the search tree (or, in a more general case, in a locally finite graph). To do that it selects the node with the minimal value of the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal. A^* is complete on locally finite graphs, path optimal if cost estimates are admissible, with an exponential time and space search cost.

We will present A^*, working as a special case of the \$-calculus $k\Omega$-optimization meta-search general procedure from previous section. For illustration, intentionally, we use the same example as in [24] with A as a root, D, F, I, J as goal states, and values of $f(n)$ written near to the states Fig. 1.

We need only three operators from the \$-calculus: cost (\$), cost choice ($\Cup$) and sequential composition ($\circ$) to describe the work of A^* (if to not count the operators of the meta-system $k\Omega$-optimization). $\Cup$ and $\circ$ are needed to express possible solutions, and \$ is used by the $k\Omega$-optimization in the examine phase to select the most promising

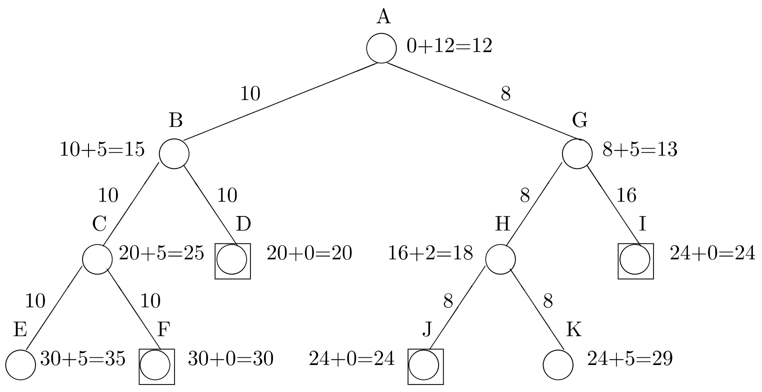

Fig. 1. A complete search tree from which A^* builds dynamically its own subtree.

node for expansion. The system consists of one agent only which is interested in everything, i.e., $\Omega = \mathscr{A} = A_1 = \{\circ, \Cup, tA, tB, tC, tG, tH, A, B, C, D, E, F, G, H, I, J, K\}$, the alphabet of simple \$-expressions $\mathscr{B} = B_1 = \{A, B, C, D, E, F, G, H, I, J, K\}$ for the LTS tree, and it uses a standard cost function $\$ = \$_1(\$_2, \$_3) = \$_2 + \$_3$, where $\$_1$ is an aggregating function in the form of addition, $\$_2$ represents costs of $k\Omega$-search, and $\$_3 = f = g + h$ represents the quality (cost) of solutions generated by A^*. Both $\$_2$ and $\$_3$ use a standard cost function from previous section. Because general choice and parallel composition are not used, thus it is irrelevant whether probabilities, fuzzy sets or rough sets were selected in the standard cost function. Strong congruence is used, i.e., *strongcong* = 1 and estimates of costs of ε (representing the heuristic h) can be different than 0. In other words, the cost of invisible actions is taken into account. The number of steps in the derivation tree selected for optimization in the examine phase $k = 1$, the branching factor $b = \infty$, and the number of steps selected for execution in the examine phase $n = 0$, i.e., execution is postponed until the optimum is found. The default goal is the minimum of \$. *gp*, *reinf* and *update* flags are reset. This means that costs of simple \$-expressions and ε cost estimates, received from the user in the *init* phase, remain fixed over loop iterations.

5.1. Optimization without uncertainty: simulation of A^*

The user provides the structure of the tree in the form of the subtrees with roots/parents A, B, C, G, H, i.e.,subtrees tA, tB, tC, tG, tH: $(:= tA\ (\Cup\ (\circ\ A\ tB)(\circ\ A\ tG)))$, $(:= tB\ (\Cup\ (\circ\ B\ tC)(\circ\ B\ D)))$, $(:= tC\ (\Cup\ (\circ\ CE)(\circ\ CF)))$, $(:= tG\ (\Cup\ (\circ\ GtH)(\circ\ GI)))$, $(:= tH\ (\Cup\ (\circ\ H\ J)(\circ\ HK)))$. This structure will be used in the *sel* phase to build incrementally the tree by applying the structural congruence inference rule from the LTS.

Only $\$_3$ is used, i.e., $\$ = \$_3$ – an optimal path is looked for, ignoring the costs of A^* search algorithm. The goal is the minimum of $\$_3$. In other words, costs of $k\Omega[t]$ \$-expression is ignored. Then consecutive steps (cycles) of the $k\Omega$-optimization simulate exactly A^* step after step (we write actions of $k\Omega[t]$ and its corresponding constructed solution \$-expression $x[t]$ instead of the cartesian product $k\Omega[t] \times x[t]$):

0. $t = 0$, initialization phase *init* from the $k\Omega[t]$ is executed: $tA = (\circ\ A\varepsilon_A)$ – the initial \$-expression is provided by the user telling that the tree tA consists of the root A and ε_A representing an invisible (not constructed yet) part of the tree.
   A is selected for expansion with the cost $(\$(tA)) = (\$_3(tA)) = (\$_3\ (\circ\ A\varepsilon_A)) = (0 + 12)$. The initial tree consists of the root state A with cost 0, an empty action ε_A representing a missing tree that contains a path to the goal with estimated cost 12.
1. $t = 1$, first loop iteration:
   Because A is not the goal state (*goal* fails and $\overline{goal}$ succeeds), the first loop iteration consisting of select, examine, and execute phases replaces an invisible ε_A one step deep ($k = 1$) by two possible children B and G using the structural congruence inference rule for the definition of tA.
   Select phase *sel*: $\varepsilon_A = (\Cup(\circ\ tB)(\circ\ tG)) = (\Cup(\circ\ B\varepsilon_B)(\circ\ G\varepsilon_G))$ is appended to an existing tree resulting in $tA = (\Cup(\circ\ A\ B\varepsilon_B)(\circ\ A\ G\varepsilon_G))$.

Examine phase *exam*: ($\$_3(\uplus(\circ\ A\ B\varepsilon_B)(\circ\ A\ G\varepsilon_G))) = \min(\$_3(\circ\ A\ B) + \$_3(\varepsilon_B), \$_3(\circ\ A\ G) + \$_3(\varepsilon_G)) = \min(10 + 5, 8 + 5)$, G is not the goal (i.e., min of $\$_3$), thus it is selected for expansion and execution is postponed ($n = 0$), and G's follow-up ε_G will be substituted by two children H and I of G in the next loop iteration.

2. $t = 2$, second loop iteration:
   Select phase *sel*: $\varepsilon_G = (\uplus(\circ\ H\ \varepsilon_H)(\circ\ I\ \varepsilon_I))$ is appended to an existing tree.
   Examine phase *exam*: ($\$_3(\uplus(\circ\ A\ B\varepsilon_B)(\circ\ A\ G\ H\varepsilon_H)\ (\circ\ A\ G\ I\varepsilon_I))) = \min((10 + 5), (16 + 2), (24 + 0))$, B selected for the expansion (more precisely, its follow-up ε_B, no pruning is performed, because A^* keeps all generated nodes in the memory).
3. $t = 3$, third loop iteration:
   Select phase *sel*: $\varepsilon_B = (\uplus(\circ\ C\varepsilon_C)(\circ\ D\ \varepsilon_D))$examine phase *exam*: ($\$_3(\uplus(\circ\ A\ B\ tC\varepsilon_C)\ (\circ\ A\ B\ D\varepsilon_D)(\circ A\ G\ H\varepsilon_H)(\circ\ A\ G\ I\varepsilon_I))) = \min((20 + 5), (20 + 0), (16 + 2), (24 + 0))$, H selected.
4. $t = 4$, fourth loop iteration:
   Select phase *sel*: $\varepsilon_H = (\uplus(\circ\ J\ \varepsilon_J)\ (\circ\ K\ \varepsilon_K))$.
   Examine phase *exam*: ($\$_3(\uplus(\circ\ A\ B\ C\ \varepsilon_C)(\circ\ A\ B\ D\ \varepsilon_D)(\circ\ A\ G\ H\ J\ \varepsilon_J)\ (\circ\ A\ G\ H\ K\ \varepsilon_K)(\circ\ A\ G\ I\ \varepsilon_I))) = \min((20 + 5), (20 + 0), (24 + 0), (24 + 5), (24 + 0))$, D selected, which is the goal and the end of the optimal path ($\circ\ A\ B\ D\varepsilon_D$) with path cost 20.
   Execute phase *exec*: the $k\Omega$-optimization stops its search, the optimal path is executed using the tree built in previous cycles, and the $k\Omega$-search re-initializes for the new problem to solve.

Note that this simulation of A^* is correct, because it simulates precisely step after step the A^* algorithm. Thus if A^* is complete and optimal, the same applies to the kΩ-optimization search (proven by Theorem 1). Moreover, the $k\Omega$-optimization may simulate many other search algorithms [14], and this represents its power and versatility.

Note that the $k\Omega$-optimization reaches the optimal solution in spite of optimizing locally one step ahead only. Note that the estimates of not expanded nodes yet, represented by the empty/invisible ε are admissible, i.e., not overestimates. Due to use of the $k\Omega$-search, the variations of A^* (new search algorithms - related to A^*, but not being A^* any more) can be considered with various values of parameters k, b, Ω and n. In particular, changing $n \neq 0$, causes that erroneous actions will be executed, and then the value of optimum for such an on-line algorithm can be different than for the classical A^* offline algorithm with $n = 0$. Note that A^* keeps all nodes in memory (no pruning) which guarantees that the optimal nodes will not be accidently removed. On the other hand, A^* may run out of memory quite fast because of the exponential space complexity. In other words, we have to take into account the costs of search too.

5.2. *Optimization with uncertainty: AU^* with probabilities*

We will call the algorithm AU^* (from A^* with uncertainty). Let us assume that in node B the robot wheels slip either making the robot to turn towards C with probability (or fuzzy set or rough set membership function) 0.4 or D with probability 0.6. In node G the robot makes turns either towards H with probability (or fuzzy set or rough set membership function) 0.3 or I with probability (or fuzzy set or rough set membership function) 0.7. These uncertainties can be modeled by the \$-calculus general choice operator The user provides the structure of the tree with in the form of the subtrees with roots/parents A, B, C, G, H, i.e., subtrees tA, tB, tC, tG, tH: $(:=tA(\uplus(\circ\ A\ tB)(\circ\ A\ tG)))$, $(:=tB(\sqcup(\circ\ B\ tC)(\circ\ B\ D)))$, $(:=tC(\uplus(\circ\ C\ E)(\circ\ C\ F)))$, $(:=tG(\sqcup(\circ\ G\ tH)(\circ\ G\ I)))$, $(:=tH(\uplus(\circ\ H\ J)(\circ\ H\ K)))$. This structure will be used in the *sel* phase to build incrementally the tree by applying the structural congruence inference rule from the LTS.

Only $\$_3$ is used, i.e., $\$ = \$_3$ – an optimal path is looked for, ignoring the costs of AU^* search algorithm. The goal is the minimum of $\$_3$. We will illustrate firstly AU^* with standard cost function using probabilities in general choice:

0. $t = 0$, initialization phase *init* from the $k\Omega[t]$ is executed: $tA = (\circ\ A\varepsilon_A)$ – the initial \$-expression is provided by the user telling that the tree tA consists of the root A and ε_A representing an invisible (not constructed yet) part of the tree.

A is selected for expansion with the cost $(\$(tA)) = (\$_3(tA)) = (\$_3\ (\circ\ A\ \varepsilon_A)) = (0 + 12)$. The initial tree consists of the root state A with cost 0, an empty action ε_A representing a missing tree that contains a path to the goal with estimated cost 12.

1. $t = 1$, first loop iteration:
Because A is not the goal state (*goal* fails and $\overline{goal}$ succeeds), the first loop iteration consisting of select, examine, and execute phases replaces an invisible ε_A one step deep ($k = 1$) by two possible children B and G using the structural congruence inference rule for the definition of tA.
Select phase *sel*: $\varepsilon_A = (\Cup(\circ\ tB)(\circ\ tG)) = (\Cup(\circ\ B\ \varepsilon_B)(\circ\ G\ \varepsilon_G))$ is appended to an existing tree resulting in $tA = (\Cup(\circ\ A\ B\ \varepsilon_B)(\circ\ A\ G\varepsilon_G))$.Examine phase *exam*: $(\$_3(\Cup(\circ\ A\ B\ \varepsilon_B)(\circ\ A\ G\ \varepsilon_G))) = \min(\$_3(\circ\ A\ B) + \$_3(\varepsilon_B),\$_3(\circ\ A\ G) + \$_3(\varepsilon_G)) = \min(10 + 5,8 + 5)$, G is not the goal (i.e., min of $\$_3$), thus it is selected for expansion and execution is postponed ($n = 0$), and G's follow-up ε_G will be substituted by two children H and I of G in the next loop iteration.
2. $t = 2$, second loop iteration:
Select phase *sel*: $\varepsilon_G = (\sqcup(\circ\ H\ \varepsilon_H)(\circ\ I\ \varepsilon_I))$ is appended to an existing tree.
Examine phase *exam*: $(\$_3(\Cup(\circ\ A\ B\ \varepsilon_B)(\circ\ A\ G(\sqcup(\circ\ H\ \varepsilon_H)(\circ\ I\ \varepsilon_I))))) = \min((10 + 5),\ (8 + 0.3 * (8 + 2) + 0.7 * (16 + 0))) = \min(15,22.2)$, B selected for the expansion (more precisely, its follow-up ε_B, no pruning is performed, because AU^* keeps all generated nodes in the memory).
3. $t = 3$, third loop iteration:
Select phase *sel*: $\varepsilon_B = (\sqcup(\circ C\ \varepsilon_C)(\circ\ D\ \varepsilon_D))$.Examine phase *exam*: $(\$_3(\Cup(\circ\ A\ B(\sqcup(\circ\ tC\ \varepsilon_C)(\circ\ D\ \varepsilon_D)))\ (\circ\ A\ G(\sqcup\ (\circ H\ \varepsilon_H)(\circ\ I\varepsilon_I))))) = min((10 + 0.4 * (10 + 5) + 0.6 * (10 + 0)),\ (8 + 0.3 * (8 + 2) + 0.7 * (16 + 0))) = \min(22,22.2)$, C and D selected. Because D is the goal and the end of the optimal path $(\circ A\ B\ D\varepsilon_D)$ with path cost 20, then the search is terminated.
Execute phase *exec*: the $k\Omega$-optimization stops its search, the optimal path is executed using the tree built in previous cycles, and the $k\Omega$-search re-initializes for the new problem to solve. However, the optimal path will be executed only 60% of time, and 40% of time node C will be reached instead (and then terminal node F that is cheaper than E).

5.3. *Optimization with uncertainty: AU^* with fuzzy sets or rough sets*

AU^* with standard cost function using fuzzy set or rough set membership functions in general choice is presented below. The fuzzy set membership function is defined in a conventional way, the rough set membership function is defined as in [23] $\mu_X^B(x) = |X \cap B(x)|/|B(x)|$.

0. $t = 0$, initialization phase *init* from the $k\Omega[t]$ is executed: $tA = (\circ\ A\ \varepsilon_A)$.
A is selected for expansion with the cost $(\$(tA)) = (\$_3(tA)) = (\$_3(\circ\ A\ \varepsilon_A)) = (0 + 12)$.
1. $t = 1$, first loop iteration:
Select phase *sel*: $\varepsilon_A = (\Cup(\circ tB)(\circ\ tG)) = (\Cup(\circ\ B\ \varepsilon_B)(\circ\ G\ \varepsilon_G))$ is appended to an existing tree resulting in $tA = (\Cup(\circ\ A\ B\ \varepsilon_B)(\circ\ A\ G\ \varepsilon_G))$.
Examine phase *exam*: $(\$_3(\Cup(\circ\ A\ B\ \varepsilon_B)(\circ\ A\ G\ \varepsilon_G))) = \min(\$_3(\circ\ A\ B) + \$_3(\varepsilon_B),\ \$_3(\circ\ AG) + \$_3(\varepsilon_G)) = \min(10 + 5,8 + 5)$, G is not the goal (i.e., min of $\$_3$), thus it is selected for expansion and execution is postponed ($n = 0$), and G's follow-up ε_G will be substituted by two children H and I of G in the next loop iteration.
2. $t = 2$, second loop iteration:
Select phase *sel*: $\varepsilon_G = (\sqcup(\circ\ H\ \varepsilon_H)(\circ\ I\ \varepsilon_I))$ is appended to an existing tree.Examine phase *exam*: $(\$_3(\Cup(\circ A\ B\ \varepsilon_B)(\circ\ AG(\sqcup(\circ\ H\ \varepsilon_H)(\circ\ I\varepsilon_I))))) = \min((10 + 5),\ (16 + 0))$. Because the membership function of I is larger than of H, I was selected in general choice. From B and I, B as cheaper has been selected for the expansion (more precisely, its follow-up ε_B, no pruning is performed, because AU^* keeps all generated nodes in the memory).
3. $t = 3$, third loop iteration:
Select phase *sel*: $\varepsilon_B = (\sqcup(\circ\ C\ \varepsilon_C)(\circ\ D\ \varepsilon_D))$.

Examine phase *exam*: $(\$_3(\Cup(\circ\ A\ B(\sqcup(\circ\ tC\ \varepsilon_C)(\circ\ D\ \varepsilon_D)))\ (\circ\ A\ G(\sqcup(\circ\ H\ \varepsilon_H)(\circ\ I\ \varepsilon_I))))) = \min(20+0{,}24+0)$, D selected. Because D is the goal and the end of the optimal path $(\circ\ A\ B\ D\varepsilon_D)$ with path cost 20, then the search is terminated.
Execute phase *exec*: the $k\Omega$-optimization stops its search, the optimal path is executed using the tree built in previous cycles, and the $k\Omega$-search re-initializes for the new problem to solve. However, the optimal path will be executed with certainty 0.6.

Adding uncertainty, increases the complexity of the optimal solution search. For illustration purposes, we decided to associate uncertainties with two nodes only. Otherwise we would have twice more nodes compared to the original A^*, i.e., every cost choice node should be followed by general choice node (something in the style of expectiminimax [24]).

For total optimization the \$-calculus can incorporate the search cost by optimizing the total cost instead of the cost of solutions only. This will use additionally a search cost function $\$_2$ and an aggregating cost function $\$_1$ in the decision process. Although the total optimization looks like much more complex than a regular optimization, in reality it is not. A small overhead (by taking care about reasoning process itself) allows automatically abandon expensive searches, because the $k\Omega$-search will avoid them as more costly. The price that we pay is that the totally optimal solutions can be of worse quality than "regular" optimal solutions.

6. Conclusions

In this paper we presented the theory for approximate reasoning based on anytime algorithms and allowing to express uncertainty in three different ways: using either probability theory, or fuzzy sets, or rough sets. The approximation of solutions is based on ideas of anytime algorithms finding incrementally better approximations of solutions in successive loop iterations of the $k\Omega$-optimization meta-search procedure. Capturing of uncertainty is possible because the \$-calculus general choice operator allows to incorporate either probabilities, of fuzzy set membership functions, or rough set membership functions. This allows potentially to experiment with various approaches to uncertainty under the same unifying framework. The same applies to the possibility to experiment with various search algorithms. The $k\Omega$-optimization allows to simulate many typical search algorithms and to construct new search algorithms.

In the paper, we presented an extension of A^* search algorithm, called AU^* using either probabilities, fuzzy sets or rough to express uncertainty. The same ideas can be incorporated to multiple agents and to the situations where both the best quality and least costly solutions are looked for. These are totally optimal solutions that automatically allow to deal with the intractability of search space.

One of the claims of probability theory is that any other methods violating axioms of probability theory (e.g., fuzzy sets or rough sets) will lead to worse payoffs as expressed by the de Finetti's theorem [4,24]. It is an open problem under which conditions this claim is valid. That claim is based on the assumption that the only correct description of the real world is by using probabilities, and not for example by fuzzy sets or rough sets.

The extension of this work could be twofold: a new axiomatization of the utility theory allowing to incorporate fuzzy sets and rough sets instead of probabilities and the standardization of the cost performance measures that may lead to a new cost paradigm of programming languages. Of course, it would be desirable to include other approaches to uncertainty, like nonmonotonic logic, or Dempster–Schafer theory of evidence. Then it would be possible truly to write that the modern *Decision Theory* = *Performance Measure Theory* + *Uncertainty Theory* [15] rather than *Utility Theory* + *Probability Theory* [24]. This is left for future research, and a current contribution can be seen as a correct step in this direction.

Acknowledgments

Comments of anonymous reviewers are appreciated. This paper is dedicated to memory of Zdzislaw Pawlak, the inventor of rough sets, who always encouraged me to work on the \$-calculus.

References

[1] L. Atallah, E. Eberbach, The \$-calculus process algebra for problem solving and its support for bioinformatics, in: Proc. 2nd Indian Intern. Conf. on Artificial Intelligence IICAI'95, Puna, India, 2005, pp. 1547–1564.

[2] L. Atallah, E. Eberbach, The \$-calculus process algebra of bounded rational agents applied to selected problems in bioinformatics, in: Proc. Third Intern. Conf. on Computational Intelligence, Robotics and Autonomous Systems CIRAS'05, Singapore, December 13–16, 2005.

[3] M. Burgin, Super-Recursive Algorithms, Springer-Verlag, 2005.

[4] B. De Finetti, Le prevision: ses lois logiques, ses sources subjectives, Ann. Inst. Poincare 7 (1937) 1–68.

[5] Ch. Duarte, G. Martel, E. Eberbach, Ch. Buzzell, a common control language for dynamic tasking of multiple autonomous vehicles, in: Proc. of the 13th Intern. Symp. on Unmanned Untethered Submersible Technology UUST'03, Durham, NH, August 24–27, 2003.

[6] Ch. Duarte, G. Martel, E. Eberbach, Ch. Buzzell, Talk amongst yourselves: getting multiple autonomous vehicles to cooperate, in: Proc. of the Autonomous Underwater Vehicles 2004: A Workshop on Multiple AUV Operations AUV'2004, Sebasco Estates, ME, June 17–18, 2004.

[7] E. Eberbach, R. Brooks, S. Phoha, Flexible optimization and evolution of underwater autonomous agents, in: N. Zhong, A. Skowron, S. Ohsuga, (Eds.), New Directions in Rough Sets, Data Mining, and Granular-Soft Computing, in: Proc. of the 7th Intern. Workshop on Rough Sets, Fuzzy Sets, Data Mining and Granular-Soft Computing RSFDGrC'99, Yamaguchi, Japan, LNAI 1711, Springer-Verlag, 1999, pp. 519–527.

[8] E. Eberbach, Expressiveness of \$-Calculus: What Matters? in: Advances in Soft Computing, Proc. of the 9th Intern. Symp. on Intelligent Information Systems IIS'2000, Bystra, Poland, Physica-Verlag, 2000, pp. 145–157.

[9] E. Eberbach, \$-Calculus Bounded Rationality = Process Algebra + Anytime Algorithms, in: J.C. Misra (Ed.), Applicable Mathematics: Its Perspectives and Challenges, Narosa Publishing House, New Delhi, Mumbai, Calcutta, 2001, pp. 213–220.

[10] E. Eberbach, Evolutionary computation as a multi-agent search: a \$-calculus perspective for its completeness and optimality, in: Proc. 2001 Congress on Evolutionary Computation CEC'2001, Seoul, Korea, 2001, pp. 823–830.

[11] E. Eberbach, P. Wegner, Beyond turing machines, The Bulletin of the European Association for Theoretical Computer Science (EATCS Bulletin) 81 (October) (2003) 279–304.

[12] E. Eberbach, Ch. Duarte, Ch. Buzzell, G. Martel, A portable language for control of multiple autonomous vehicles and distributed problem solving, in: Proc. of the 2nd Intern. Conf. on Computational Intelligence, Robotics and Autonomous Systems CIRAS'03, Singapore, December 15–18, 2003.

[13] E. Eberbach, D. Goldin, P. Wegner, Turing's Ideas and Models of Computation, in: Ch. Teuscher (Ed.), Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, 2004, pp. 159–194.

[14] E. Eberbach, A. Eberbach, On designing CO\$T: A new approach and programming environment for distributed problem solving based on evolutionary computation and anytime algorithms, in: Proc. 2004 Congress on Evolutionary Computation CEC'2004, vol. 2, Portland, OR, 2004, pp. 1836–1843.

[15] E. Eberbach, Decision theory = performance measure theory + uncertainty theory, in: Proc. 10th Intern. Conf. on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing RSFDGrC'2005, Regina, Canada, LNAI 3641, Springer-Verlag, Part I, 2005, pp. 551–560.

[16] E. Eberbach, \$-Calculus of bounded rational agents: flexible optimization as search under bounded resources in interactive systems, Fundamenta Informaticae 68 (1–2) (2005) 47–102.

[17] E. Eberbach, Capturing evolution of polymorphic viruses, in: Proc. IX National Conf. on Evolutionary Computation and Global Optimization KAEiOG'06, Murzasichle, Prace Naukowe, Elektronika, z.156, Politechnika Warszawska, Warsaw, Poland, 2006, pp. 125–138.

[18] E. Eberbach, Expressiveness of the π-calculus and the \$-calculus, in: Proc. 2006 World Congress in Comp. Sci., Comp. Eng., & Applied Computing, The 2006 Intern. Conf. on Foundations of Computer Science FCS'06, Las Vegas, Nevada, 2006, pp. 24–30.

[19] P.E. Hart, N.J. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Trans. SSC SSC-4 (1968) 100–107.

[20] E. Horvitz, S. Zilberstein (Eds.), Computational tradeoffs under bounded resources, Artificial Intelligence 126 (2001) 1–196.

[21] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, I & II, Information and Computation 100 (1992) 1–77.

[22] Z. Pawlak, Rough sets, International Journal of Computer and Information Science 11 (1982) 341–356.

[23] Z. Pawlak, A. Skowron, Rough set rudimentals, Bulletin of International Rough Set Society 3 (3) (1999) 181–185.

[24] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995, 2nd ed. 2003.

[25] P. Wegner, E. Eberbach, New models of computation, The Computer Journal 47 (1) (2004) 4–9.

[26] J. Von Neumann, O. Morgenstern, Theory of Games and Economic Behavior, Princeton University Press, 1944.

[27] L.A. Zadeh, Fuzzy sets, Information and Control 12 (1965) 338–353.