MODELLING THE STRUCTURED AND COMPLEX DECISION SITUATIONS

Martin Hrubý[1]

[1]Brno Technical University, Faculty of Information Technology
Bozetechova 2, Brno, 61266, Czech Republic

hrubym@fit.vutbr.cz(Martin Hrubý)

Abstract

The paper deals with modelling and simulation of complex intelligent systems where the modelled entities have to make certain decisions in conditions of for example competitive market. We base this paper on our experiences in making models of competitive markets with electricity and other auxiliary services in area of Czech republic and Central Europe. The studied problem is certainly an excellent example of multi–player competition on multi–market with multi–commodity (a player has to decide what commodity is going to offer at which market in competition to other players). In this paper, we are more focused on theoretical analysis of the problem and its effective implementation in simulation models. We transform these situations of strategic decisions of one or more intelligent entities to so called structured and complex decisions. We are going to present a rather general solution to these systems built from elementary decisions with a special emphasis on their effective simulation. We use a special mathematical formalism called Automated Information Net (AIN) to model relations between elementary parts of the AI model. AIN can predict these computing iterations which would not bring new information to simulation of our models. We save some simulation time by this principle. As our background is mostly in area of economics modelling, we oftenly use market and production examples to demonstrate rather theoretical problems. The use of our ideas is hopefully wider.

Keywords: Modelling and simulation, game theory, decision making, complex systems, market models

Presenting Author's Biography

Martin Hrubý[1] (born 1976 in Olomouc, Czech Republic) received his MSc. in Computer science in year 2000 and Ph.D. in year 2004. His Ph.D. thesis was oriented to heterogeneous modelling and simulation. Last few years he is more focused to artificial intelligence and mathematical game theory. His professional interest is in bringing theoretical disciplines like game theory to practical use. As a coauthor, he cooperated in research and development of models of market with electricity and auxiliary services. He uses his practical experience in modelling AI to formulate more general conclusions. He is also interested in GIS systems and computer games. He speaks english and spanish.

1 Introduction

In our recent work we have been modelling intelligent behavior of producers and traders in rather large and complex market with electrical power and so called ancillary services [3]. After many years of development, the models became too complex for further maintenance and development, so we had to start thinking about their optimization and easy reconfigurable programming [5].

The mentioned models expresses decisions of intelligent entities in conflict (competition) or cooperation with other intelligent entities. This situation is well known as a *game situation*, the intelligent entities are usually called *players* and the complete mathematical apparatus describing games is called *the game theory* [1, 2]. One can find a plenty of theoretical papers [6] about modelling these problems, but in our best knowledge there is no public record about a practically implemented model. There is also a theory called the *decision theory* which is probably more general than terms discussed here. Anyway, we treat the decisions by meaning of artificial intelligence like task solving, playing games, searching in state space and so on.

Having a model of an intelligent behavior has many contributions and benefits: by experimenting with such a model, we can better understand the reasons why the modelled entity does what it does. At second (and this is probably more utilizable), we can so *predict future behavior* of the entity or of a group of them. In case of market models, the future prices and market events are very interesting for many people and institutions. The main aim of this paper is to start a series of more advanced analysis of decision making which comes as a generalized view to our practical experiments and experience in such a modelling.

There is a very important assumption and agreement in the beginning and it is that we always prefer *numerical (simulation) solution* to our situations instead of the analytical one. Let us quickly define what we mean by these two modelling approaches: in analytical modelling, we attempt to describe a system at very abstract and general level using pure mathematical inference rules. As a result we obtain a very simple and powerful mathematical law in form of (for example) mathematical formulas. We input some parameters and get the result. Formulating such a law is probably very difficult intellectual task and mostly impossible. One may also agree that this approach is not very flexible to changes of specification. On the other hand, we have methods of numerical solution where basically numerical numerical experimenting method substitutes the analytical reasoning by machine iterative processing.

So, doing models in numerical manner require certain number of computational steps to finish the size of the task. We are going to show that number of steps grows with $|S|^Q$ where S shows a level of detail in our model and Q inputs a dimension of the problem. This time complexity might be extreme in practical model applications and any attempt to decrease number of evaluated iterations is welcome. We published one possible way in [5] and we are going to extend this result with more wide approach in structured decisions.

In the beginning of the paper, we will define elementary and structured decision problems which are the the topics of our main interest.

2 Introduction to decision problems

When talking about thinking and decision making, we have to say that in the current state of AI machines (and this will not be probably any surprise) *cannot think* by the human meaning of that expression, but they can *reasonably choose* one of some predefined actions (e.g. predefined by their designer). This approach somehow follows our human way of analytical thinking when the human decomposes the situation to set of possible actions (opinions) and analyses how each action will possibly change his current state. The difference of new and previous states is then his *profit* caused by choosing the action. So, by saying that the machine does some decision, we always mean that it somehow chooses one of the predefined actions. The next step in AI development would be probably a machine able to formulate itself independently a language of its actions to choose from.

We are going to study this area in more mathematical manner. So, we start with a definition of *a domain* of the problem, definition of *a decision* itself and their *classification.*

Domain

Let us have a situation when an intelligent entity has to decide what strategy he/she is going to adopt in problem with domain denoted by:

- a discrete set of actions $D_d = \{a_1, a_2, ...\}$, or
- by a continuous interval $D_c = \langle f, t \rangle$.

Decision

The entity must be able to evaluate each action of his domain (D is D_c or D_d type) by some profit using the *utility function* which we are going to denote by $u : D \to \mathcal{U}$, where $\mathcal{U}$ is some universum of all possible profits (but usually we work with numbers). We also expect that the profits are comparable, so that there is some binary relation $\leq$ on $\{u(a) | a \in D\}$, so it is possible to compare if a_1 is better then a_2. The entity will probably choose $a^* \in D$ with the highest utility $u(a^*)$ and this a^* is then *the solution* to our decision situation. The solution a^* represents some *problem optimum* for the entity (this term is more clear in context of multiplayer games).

As we already mentioned in the introduction, we always prefer *simulation approach* to the problem instead of analytic and so, discretizing the problem is necessary – if working with naturally continuous problem, we have to transfer the continuous interval D_c to a discrete set $S = \{s_1, s_2, ..., s_n\}$. Let us use a term *strategy* instead

Map of classes	single (players = 1)	multiple (players ≥ 2)
elementary problem	$C1$ (easy)	$C3$ (game theory)
structured problem	$C2$ AIN	$C4$ AIN $\times$ game theory

Fig. 1 Map of decision situations with complexity classes $C1 < C2 < C3 < C4$

of *action* from now. The solution is again some strategy $s^* \in S$. The solution s^* is preferred, when there is no other $s \in S$ for which $u(s) > u(s^*)$. There might be $s \in S$ such $u(s) = u(s^*)$. For these we say, that the entity is indifferent between them and he/she chooses randomly from all $S^* = \{s \in S | u(s) = u(s^*)\}$ (to equilibrium becomes so called *mixed*). But we do not study such situation in this paper. Anyway, methods of choosing equilibria within state space of strategies is not an issue in this paper.

Classification

Well, finding the highest $u(s)$ is rather easy. We are going to make it more difficult by adding these dimensions/views/problems to the decision situations:

- to do it quickly – we certainly want to model our problems effectively, so that the machine interpreting them can react instantly. Real solved problems have an extreme time complexity.
- to let some more intelligent entities to attend the situation (in meaning of game theory) – decision situations becomes more interesting when other entities can affect mutually their profits. Choosing a strategy from the state space is not as trivial like in previous situations. There is a rather large theory of games describing the strategic thinking above the state space of all possible actions of me and my opponents.
- to do the decision structured – we are going to study situations when one decision is based on our conclusion from another decision. The decisions then create a hierarchy or a sequence.

The Figure 1 demonstrates all possible problem classes in meaning of number of players and level of problem and their solving. In our experience, these classes can be ordered by their implementation difficulty. The easiest class is $C1$ which also demonstrate the pure principle of the AI methods of decision making. $C2-$class is fundamentally similar to $C1$, just the strategy set is multidimensional or the utility function recursively calls inner decisions (we will show $C2$ made by AIN approach). $C3$ is harder then $C2$ as the state space grows with number of players and we have to start using some optimizations to its evolution. Other reason for putting $C3 > C2$ expresses its scientific problems in game equilibrium definition (Nash, Stackelberg, correlated, ...) and so on. $C4$ is on the top of complexity. $C4$ is the class where we recommend using AIN technique to optimize the model structure and simulation time necessary to do simple experimenting with a model. AIN is a method developed in [4], now used to optimize problems in decision modelling.

3 Elementary decision problems

We define the decision problem to be $\Pi = (S, C, u)$, where:

- S is a *final* set of strategies (actions) which define what choices a player has,
- C is a *context* of the decision (set of objects) – this information describe for example internal attributes of a player or specific rules in a decision situation.
- $u(s)$ is a *utility function* which gives an utility (or profit) to each $s \in S$.

And the solution is some sort of

$$s^* = arg[max_{s \in S}\{u(s)\}] \quad (1)$$

In an implementation point of view to the problem, we model the utility function as a mathematical function of strategy $s \in S$ and decision context C:

$$u(s) = function(s, C) \quad (2)$$

3.1 C1–class: the introduction

Algorithm 1 C1–class: Simple decision algorithm

```
res = {}
for s in S:
   res[s] = utility(s, C)
solution = res.max()
```

The time complexity of Alg. 1 is mostly done by complexity of enumerating the $utility(s, C)$ function and, of course, by its $|S|$–times repetition and finding the highest computed utility (also linear). Optimizing the "utility" function is required.

We may extend this concept in many varieties, like finding the first good–enough solution, or, if there is more than one equally ranked solutions, we choose then randomly among them. Again, selecting the right solution is not the topic of this paper.

Definition (elementary decision): Let $\Pi = (S, C, u)$ is a decision problem. If its context C contains only constant objects, then Π is elementary. Hopefully it is clear what we mean by *an object* in terms of computer science. Object $a = 3$ is a constant, object Π is a decision.

price s	10	20	30	40
profit	200	1200	2200	3200

Fig. 2 C1–class situation. Profit made by $utility_1(s, 100)$

3.2 C3–class: Game theory involved

The game is defined as

$$\Gamma = (Q; S_1, S_2, ...S_N; u_1, u_2, ..., u_N) \qquad (3)$$

where Q is set of $|Q|$ number of players, $S_i; i \in Q$ are strategy sets and u_i are utility functions of players $i \in Q$.

Elementary decision becomes more complex when there are more decision makers who compete each other. There are two basic approaches that we have to implement in our model:

1. Utility function of a player is not only a function of his parameters but also of his colleagues/enemies. Model of utility function becomes more complex. But generally, it is still some $function(s, C)$ of context C and profile $s \in S$ ($|s| \geq 2$). Final form of existence of utility function is an implementation detail. Saying that there are $|Q|$ utility functions u_i should emphasize that each player has generally *different* payoff in certain profile $s \in S$.
2. State space of the situation grows with number of players. Generally there are $|S_1 \times S_2 \times ... \times S_{|Q|}|$ items (cells).

Let us demonstrate multi–player decision making (game theoretical) by a simple market situation with constant demand of 100 items. We are starting with a single–player in form $C1$.

Player p_1 has fixed production cost $fc_1 = 5$ per item and variable cost $vc_1 = 3$ per item. His production capacity is $cap_1 = 100$. Selling t_i items he makes a profit:

$$utility_i(s_i, \{t_i, vc_i, cap_i, fc_i\}) = t_i \cdot (s_i - vc_i) - cap_i \cdot fc_i \qquad (4)$$

We can see (Figure 2) that the player is free to choose any strategy from $\{10, 20, 30, 40\}$, surely the $40-$strategy, or higher if possible until some form of elasticity demand would start decreasing the demand.

What about the situation if two players are involved and consumers behave that:

- they buy first from the cheaper player and if all is sold out, then they buy the rest from the more expensive player,

s_1 / s_2	10	20	30	40
10	$-150, -1200$	$200, -900$	...	$200, 100$
20	$350, -1200$	$350, -200$	...	$1200, 100$
30	$850, -1200$	$850, -200$	...	$2200, 100$
40	$1350, -1200$	$1350, -200$	...	$1350, 1800$

Fig. 3 C3–class situation. Profit made by $utility_i(s, X)$

- if both play the same price strategy, consumers buy randomly with uniform distribution, so every player i with an offer o_i sells $\frac{o_i}{o_1+o_2} \cdot demand$, if $demand < o_1 + o2$, otherwise they both sell all.

Let us see the second situation with a second player p_2 having $cap_2 = 200$, $fc_2 = 8$ and $vc_2 = 6$.

A now begins the complicated way of analyzing the game matrix and finding the solution. Our question is: what strategy will players p_1 and p_2 choose in this situation? Both have to decide somehow.

Again, the model of such a decision is a sequence of computing the state space and its analysis.

Algorithm 2 C3–class: Game decision algorithm

```
res = {}
for s1 in S:
   for s2 in S:
      res[(s1,s2)] =
          utility_1((s1,s2), C)

solution = res.gametheory()
```

An example of implementing the utility function is described more in details in section 6. Also, finding optimum $res.gametheory()$ is a job for game theory methods and is not an issue in this moment.

4 Introduction to AIN

In this section, the AIN method will be introduced. AIN is the core to this paper and AIN is used to organize the highest level of decision problems ($C3, C4$).

AIN (Automatic Information Net) is a modelling method originally developed for modelling of heterogeneous systems [4]. During development of market models we noticed that the AIN method can by applied also in game–theory models, especially to organize structure of models and to do automated optimizations during the simulation run–time.

The principle of AIN lies in construction of a net $[OS, R]$ (oriented graph) of OS objects and their interconnections $R \subseteq OS \times OS$. An edge $r = (o_1, o_2), r \in R$; expresses that o_2 derives its value from the value of o_1–object, so

$$o_2 = function(..., o_1, ...) \qquad (5)$$

Computing in AIN (evolution) stands on continual keeping the system of OS objects mutually consistent through their relations. So, if any object $o \in OS$ changes its value, then all $o\bullet = \{x \in OS | (o,x) \in R\}$ (poset of o) must be automatically re–evaluated ($\bullet o$ denotes a preset of o).

The AIN evolution context is an tuple $AC = (M, CH)$. The evolution $AC_1 \vDash AC_2$ periodically turns the AIN system between so called *marked state* ($M \subseteq OS, CH = \emptyset$) and *changed state* ($M = \emptyset, CH \subseteq OS$).

More formally, one evolutionary step is transforming the AIN–context $AC = (M, CH)$ to $AC' = (M', CH')$, so that:

1. In the *marked state*, all $o \in M$ are reevaluated and if o changes its value, it is then inserted to the CH set.
$$CH' = \{ch \in M | reeval(ch) = true\}; M' = \emptyset$$
2. In the changed state, all $\bigcup_{o \in CH} o\bullet$ are inserted to the M set.
$$M' = \bigcup_{o \in CH} o\bullet; CH' = \emptyset$$
3. If $M = CH = \emptyset$, the system becomes stable and the evolution terminates.
$$M' = \emptyset; CH' = \emptyset$$

More formal and rigorous definition could be found in [4] where we describe also sequential (processes) part of the AIN specification.

Algorithm 3 Simple example of an AIN-model

```
a=2
d=3
b=a+1
c=b+d
```

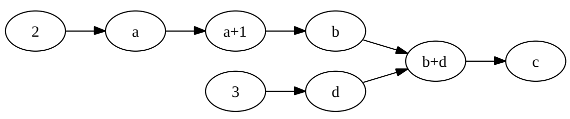


Fig. 4 AIN model for the example

The example (Algorithm 3) has its graphical form in Figure 4 and the computation evolves as follows:

$$(\{2,3\},\emptyset) \vDash (\emptyset,\{2,3\}) \vDash$$
$$(\{a,d\},\emptyset) \vDash (\emptyset,\{a,d\}) \vDash$$
$$(\{a+1,b+d\},\emptyset) \vDash (\emptyset,\{a+1\}) \vDash$$
$$(\{b\},\emptyset) \vDash (\emptyset,\{b\}) \vDash$$
$$(\{b+d\},\emptyset) \vDash (\emptyset,\{b+d\}) \vDash$$
$$(\{c\},\emptyset) \vDash (\emptyset,\{c\}) \vDash$$
$$(\emptyset,\emptyset)$$

In the beginning, only the objects o with $\bullet o = \emptyset$ like $\{2,3\}$ are marked. The system evolves until it reaches a stable state when no object reacts on someone's change – so, from some context the system evolves $(\emptyset, ch) \vDash (\emptyset, \emptyset)$. This is the final state of the evolution.

Obviously, evolution of a particular AIN-system does not terminate if that system contains some bad cycle. But this is possible in any language. Normally the AIN-evolution terminates after proceeding of final number of evolutionary steps. Effect of here presented optimization also assumes that the system gets stabilized in (much) less steps than without control of AIN (so that really saves some simulation time).

5 Applying the AIN in basic problems

Let us have a look on Figure 5 with an empty game matrix of $(\{A,B\}; S_A, S_B; U_A, U_B)$, $S = S_A \times S_B$. All cells of the matrix have to be evaluated with profits $U(S)$ so, that in the next phase we could use game theory analysis to find the required solution. We call $cellModel$ the general algorithm which computes the utility for all $s \in S$.

A/B	s_1	s_2	...	s_K
s_1				
s_2				
...				
s_K				

Fig. 5 Game matrix for two players game

We express the $cellModel$ as an AIN-system $cellAIN = [OS_c, R_c]$ with objects of global and local meaning. Objects s_A, s_B will hold current row (A–player) and column (B–player) strategy, objects u_A, u_B will be filled with currently evaluated profits of the players. So, $s_A, s_B, u_A, u_B \in OS_c$. To be more formal, we expect $\bullet s_A$, $\bullet s_B$, $u_A\bullet$ and $u_B\bullet$ to be empty sets (see Figure 6).

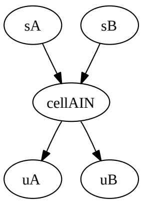


Fig. 6 AIN system for iterating

We are now going to describe two methods of application of AIN (so called Simple iterating and Global instances) and two other related notes about possible model persistence and parallelization.

5.1 Simple iterating

In the simple iterating, the $cellAIN$ sweeps across the matrix as illustrated in the following algorithm.

Algorithm 4 Simple iterating

```
forall $s_A$ in $S_u$
 forall $s_B$ in $S_u$ {
 start evolution;
 $U_1(s_A, s_B) := u_A$;
 $U_2(s_A, s_B) := u_B$;
 }
```

The evolution starts with $(\emptyset, \{s_A, s_B\})$ and after certain number of steps it terminates. Then, u_A and u_B contains evaluated profits.

Let S_q is a sequence of unrolled state space S. Then c_i denotes number of computing steps done when computing the $i-$the profile in the S_q sequence. It is obvious, that number of steps will differ between each two cells c_i and c_j (this will be shown in the following case study), but in summary will

$$\sum_{i \in \{1,\ldots,|S_q|\}} c_i << c_1 |S_q| \tag{6}$$

5.2 Global instances

This approach leads probably to the best results and the effectiveness of the simulation grows with complexity of $cellAIN$ model (when reduced number of steps do its effect).

The constructed AIN-system consists of $|S_A \times S_B|$ instances of $cellAIN$, it means that each cell c (with a corresponding strategy profile $s^*_c \in S$) of game matrix has its own instance of $cellAIN_{s^*_c}$ model connected to a corresponding vector of strategy cells s^*_c. The evolution starts from $(S_A \cup S_B, \emptyset)$ context. All particular operations in $cellAIN$s are now evaluated simultaneously. We enclose a practical case study demonstrating this principle.

5.3 Persistence

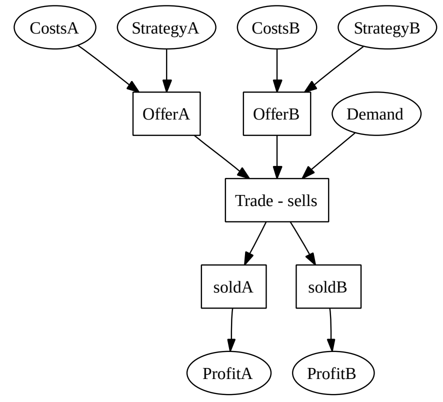
This representation of a model is ideal to make the model persistent. It means, that the computer program interpreting the AIN–system can export and save $[OS, R]$ and re–load it back next time when executed. In fact, the context of re–loaded AIN-system would be $(\emptyset, \emptyset)$. Then the particular experiment which modifies $e \subseteq OS$ can be computed in much less steps then the whole system $[OS, R]$. Persistent existence of objects is a basic feature of language like Smalltalk and Self. Unfortunately, these languages are not so fast in executing their programs, so we have to use C/C++.

5.4 Parallelization

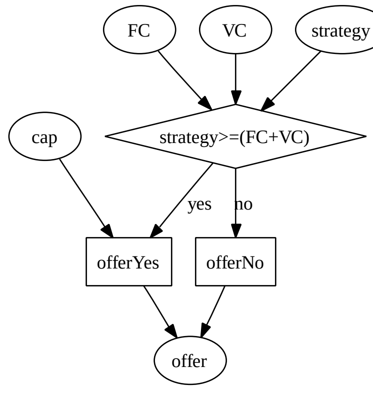
Other contribution to making the simulation even more efficient is to make the *marked step* in parallel. It means, in $AC = (M, \emptyset)$ state, we can split the M–set to as many subsets $M_1, M_2, ..., M_{cpus}$ we want (where $cpus$ is a number of CPUs available). The marked state is then evolved as follows:

Algorithm 5 Parallel execution of the marked state

```
$(M_1, M_2, ..., M_{cpus})$ = split($M$)
create $cpus$ threads $i \in \{1..cpus\}$:
 $CH'_i = markedstate(M_i)$
$CH' = \bigcup_i CH'_i$
```



a)



b)

Fig. 7 a) Main view on the model ($cellModel$) b) Model of offer

6 Experimental case study

Let us have an example of a primitive oligopoly market with a price competition, where the $cellModel$ is defined in Algorithm 6 and AIN-like illustration on Figure 7.

The system is defined using parameters which should be rather clear: variable costs (vc_i), fixed costs (fc_i), demand, production capacities (cap_i).

Algorithm 6 Simple market model

```
profit_i=receipts_i-costs_i
receipts_i=strategy_i*sold_i
costs_i=vc_i * sold_i + fc_i
sold_i=sells(demand,
   [(amount_1,strategy_1),
    (amount_2,strategy_2),...])
amount_i=offer(strategy_i, vc_i,
    fc_i, cap_i)
```

Cells	Building time	Initial calc. time	Reeval steps	Index reevals
100	0,00988	0,00014	880	"X"
200	0,02171	0,00059	1720	1,95
400	0,03768	0,00142	3360	1,95
800	0,07509	0,00283	6640	1,98
1600	0,14496	0,00557	13120	1,98
3200	0,30989	0,01515	26080	1,99
6400	0,71796	0,02629	51840	1,99
12800	1,77498	0,04839	103360	1,99
25600	4,98704	0,09239	206080	1,99
40000	10,3814	0,14230	321600	1,56

Fig. 8 Model building times ($steps_0$)

The model is constructed in computer memory respecting the method "Global instances" and the AIN initiates the first evolution process. We made a prototype model for purpose of this paper. See Figure 8 with measured experimental times of model initialization. The table 9 shows sequence of experiments (referred to 160 × 160 matrix, 25600 cells) and response time to particular inputs. At first, the model is initiated. Then, we change the global variable $demand$ and the whole simulation is recomputed in 81% of original time. Other experiments goes on. When we input a new fixed cost of A–player, the whole run takes only 0.25 of time.

6.1 Computing the game equilibrium

We have not spoken about computing the final results yet – it means, about solving some sort of stable point of the game system. There are many possible interpretations of game equilibrium. They all are mentioned in theoretical research papers of game theory and in the game-theoretic literature.

Step	Change	Reevals	$\frac{steps}{steps_0}$
0	Initial	206080	X
1	Demand = 200	166641	0,81
2	FC0 = 20	51361	0,25
3	VC1 = 4	52641	0,26
4	CAP0 = 100	153921	0,75
5	Demand = 400	165204	0,8

Fig. 9 Reaction time to the experiments (160 × 160 matrix)

In the applied point of view, as we do in our research, we understand this last computation phase to be another AIN-system connected to the "matrix AIN-system". They both make a pair (matrix, equilibrium solver) which is some $[OS_p, R_p]$ where $eq \in OS_p$ is an object containing the resulting information (it points to the equilibrium profile).

When doing heterogeneous or inner games, we may interconnect them in this way. For example, equilibrium object eq of some game may be included in preset of $cellModel$ of another game. The whole system then works as a complex.

7 Structured decision problems

At the beginning, the most simple view on structured decisions is that we have to choose one action from strategy set S_1 and one from S_2, so we have to make two decisions and these two correlate. We can transform this situation to choosing from strategy set $S = S_1 \times S_2$, to a decision $\Pi = (S_1 \times S_2, C, u)$. This approach is not very efficient for many reasons and we rather prefer a two step decision (where the order of steps is important) – for example $\Pi_1 = (S_1, C_1, u_1)$ and $\Pi_2(S_1) = (S_2, C_2, u_2); S_1 \in C_2$ with:

$$u_2 = function_2(s_2, C_2 \cup \{S_1\})$$

$$u_1 = function_1(s_1, C_1 \cup \{\Pi_1(result)\})$$

Let us start thinking about structured problems, so about problems where one decision is composed of other decisions – the context of decision is not constant and includes results of other decisions and the sub-decisions make a tree hierarchy.

We may recognize situations where:

- context is constant for the whole decision, so that we can make this decision before the main decision process. Symbolic transcription: $A_1, ..., A_n \Rightarrow B$.
- context is parameterized by the s strategy profile of the main decision (has to be instantiated for each cell), so that the evaluation of sub-decision is a part of evaluation of each profile. Symbolic transcription: $B(A_1(s_B), ..., A_n(s_B))$.
- there is a mixture of constant and parameterized sub-decisions. The most general situation. Symbolic transcription: $A_1, ..., A_n \Rightarrow C(B_1(s_C), ..., B_n(s_S))$.

Implementation of these three possibilities is probably clear. We would like to show an implementation made by AIN approach showing all derived contribution, mainly in modelling point of view and efficient simulation view.

8 AIN implementation of the decision

Let us have an $N-$ player game situation $\Gamma = (Q; S_1, S_2, ...S_Q; U_1, U_2, ..., U_Q)$, S denotes set of strategy profiles and $cellModel(s, C); s \in S$ is a some sort of model of utility functions U_i.

We are going to transfer such a specification to AIN system $\Sigma = [OS, R]$. The system Σ will contain:

1. AIN objects for all strategies. Let $S_u = \bigcup_i S_i$.
2. AIN objects for all objects of context C.
3. AIN objects (subnets) reprezenting cellModels. $Cells = \{cellModel(s, C) | s \in S\}$.
4. AIN object Σ_r keeping *a result* to the decision. This object is responsible for computing some form of equilibria.

Σ_r reprezents a result to Σ decision situation. We expect that Σ_r objects may be shared with other decisions. Set of objects SO is finally composed of:

$$OS = S_u \cup C \cup Cells \cup \{\Sigma_r\}$$

Final interconnection of objects inside the OS set depends on particular $cellModel$, but very generally R would contain some subset of:

1. $R_s = \{(s_i, cellModel(s, C)) | s = (s_1, s_2, ..., s_{|Q|}) \in S, 0 \le i \le |Q|\}$
2. $R_c = \{(c, cellModel(s, C)) | s \in S, c \in C]\}$
3. $R_r = \{(s, \Sigma_r) | s \in S\}$

$$R' = R_s \cup R_c \cup R_r$$

Resulting R will be a minimal working subset of R'. Static minimalization of R is a topic for another research. Size of R determine a number of objects which will be marked for reevaluation during the very step of AIN evolution. Also construction of $cellModel$ is extremely important for effectiveness of computation.

8.1 Composition: $A_1, ..., A_n \Rightarrow B$

Set of mutually independent decisions $A_1, A_2, ..., A_n$ which have their AIN equivalents in set of AINs $\Sigma_1 = [OS_1, R_1], \Sigma_2, ..., \Sigma_n$ will make global resulting AIN:

$$\Sigma_{As} = [OS_1 \cup OS_2 \cup ... \cup OS_n, R_1 \cup R_2 \cup ... \cup R_n]$$

with decision objects $\{\Sigma_{r1}, \Sigma_{r2}, ...\Sigma_{rn}\}$.

If this set of decisions should be presumption for B ($A_1, ..., A_n \Rightarrow B$), where $B = (S_B, C_B, cellModel_B)$, $\{\Sigma_{r1}, \Sigma_{r2}, ...\Sigma_{rn}\} \subseteq C_B$then we obtain $\Sigma = \Sigma_{As} \cup \Sigma_B$.

8.2 Composition: $B(A_1(s_B), A_2(s_B), ..., A_n(s_B))$

This type of structured decision solves $B = (S_B, C_B, u_B)$ where computing of u_B depends on decisions $A_1(s_B), A_2(s_B), ..., A_n(s_B)$ made in context of profile $s_B \in S_B$.

s_B	s_{b1}					s_{bn}
cell		s_{A1}	s_{A1_1}	s_{A1_n}		
		u_{A1}	$u_{A1}(s_{b1})$			

The AIN system Σ_B will be constructed as it was described above. Just its $cellModel$ reprezenting u_B for all $s_b \in S_B$ has to include *an instance* (a clone) of composition of $A_1, ..., A_n$, all having s_b in their context, so an edges from s_b to $cellModel_{Ai}$ will be included in resulting AIN.

8.3 C4–Class

AIN implementation of $C4$–class decision problem is a composition of AIN implementation and game approach. It means that all decisions are results of game conflict and the profile variable $s \in S$ is a vector of pure strategies of all players in the game. The rest is similar to methods described above.

9 Conclusion and future work

In this paper, we have presented a method of modelling complex decision problems. We recognized decisions in situation where there is only one decision maker involved ($C1$) or situations with more decisions makers ($C3$) where the methods of game theory must by employed. The main result of the paper is in efficient modelling and simulation of more advanced decision situations ($C2$, $C4$) with structured problems when making one decision is dependent on making another decision. It is even more difficult with more player, it means in game theory situations.

The presented theory is not finished and is still in development. We implemented a very efficient AIN kernel and few demonstration models based on the described method. Everything is promising that contributed approach leads to a very efficient modelling of multiplayer structured decisions and mainly to their fast executing on PC computers and experimenting with them.

Acknowledgment: This work has been supported by the Grant Agency of Czech Republic grant No. GP102/06/P309 *"Modelling and Simulation of Intelligent Systems"* and the Czech Ministry of Education under the Research Plan No. MSM 0021630528 *"Security-Oriented Research in Information Technology"*.

10 References

[1] Aumann, R. J.: Game Theory, *The New Palgrave: A Dictionary of Economics*, Volume 2, edited by

J. Eatwell, M. Milgate, and P. NewMann, pp. 460-482, Macmillan, London, 1987

[2] Myerson, R. B.: Game Theory: Analysis of Conflict, Harvard University Press, 2004

[3] Hruby, M., Toufar, J.: Modelling the Electricity Markets using Mathematical Game Theory, In: Proceedings of the 15th IASTED International Conference on Applied Simulation and Modelling, Rhodos, 2006, CA, s. 352-357, ISSN 1021-8181

[4] Hruby, M.: Formal Specification of the HELEF Simulation Language, In: Proceedings of 37th Internation Conference MOSIS'03, Ostrava, CZ, MARQ, 2003, s. 143-148, ISBN 80-85988-86-0, 2002

[5] Hruby, M., Bednar, J.: Automated Optimizations of Models Based on Game Theory, In: Proceedings of INISTA 2007, Istanbul, Turkey, 2007

[6] Song, Y. et all: Analysis of Market Power in Oligopolistic Electricity Market Based on Game Theory, *Proc. Power Systems and Communications Infrastructures for the Future*, Beijing, 2002