Synthesis of static situative structures. Mathematical model

Alexander Kosorukoff and Eugene Panteleev

12/27/1993

Abstract

All activity of a man is devoted to constructing desirable relationships between material and formal objects. In this work, an approach for solving a narrower task is suggested: the synthesis of desirable relationships between objects presented as black boxes with some inputs and outputs. The existence of the relations is usually restricted by some conditions. These conditions defined either locally (the rules of connectivity between inputs and outputs) or globally (the criteria of existence of the whole structure, the set of objects and relationships). The applications of this method range from tasks of barter exchange optimizing (based on a set of commercial offers) to tasks of constructing educational programs out of basic discipline modules.

1 Introduction

All activity of a man is devoted to constructing desirable relationships between material and formal objects. In this work a mathematical approach for solving a narrower task is suggested — synthesis of desirable relationships between objects that are presented as *black boxes* with their sets of *inputs* and *outputs*. The conditions of existence for each relationship between objects are defined on two levels: either local (the rules of connectivity between inputs and outputs) and global (the criteria of existence of the whole set of objects and relationships involved). The structures constructed of these primitives will further be designated as static situation structures.

The application area of this method is wide enough: from tasks of barter exchange optimizing based on a set of commercial offers to tasks of constructing educational programs out of basic discipline modules.

2 The model

Assume we have some set of economical subjects. Each of them can be presented as a black box with input vector IN() and output vector OUT(). These vectors are respectively demand and supply of the subject (Fig. 1).

The local condition of connectivity is clear: supply must satisfy demand. The global criterion is the absence of a subject in a structure with unsatisfied demand. In other words all subjects involved in the exchange must be at least satisfied. Some of them can get more than expected, but not less. Also here we have a criterion to compare two structures of exchange and decide which of them is better. It's the measure of satisfaction of all subjects. It is possible to restrict the number of modules involved to avoid unreliable structures with too many subjects. We can construct other useful global criteria, but not here to prevent us from making the task more complex.

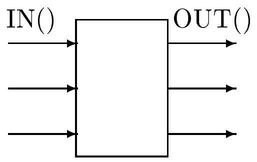

Figure 1: Module definition

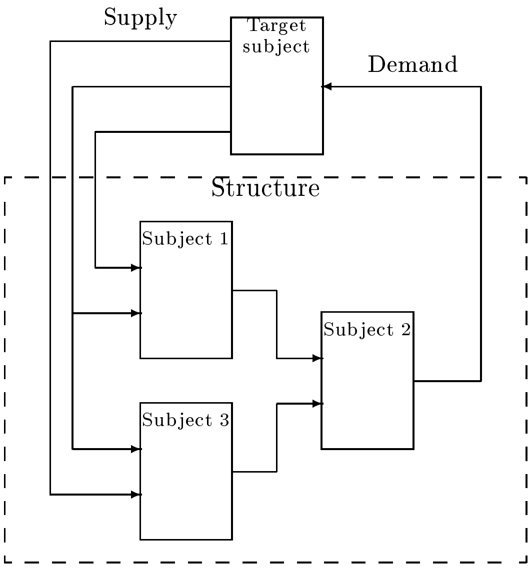

Figure 2: Structure of exchange for a given subject

The result of the synthesis is a set of possible static structures short-circuited on a target subject (Fig. 2).

Let us use a set theory and first order predicate logic to formalize the task. All varieties of goods that can form supply or demand is designated as $G = \{x \mid x \text{ is a kind of goods}\}$. The number of elements of set G is designated as #(G). The set of subjects is designated as S. When person P changes his automobile for money, it can be presented as following logical predicate:

```
Change(A,{money},{automobile})
```

This predicate is logical function:

$$Change:\ S \times 2^G \times 2^G \rightarrow \{true,\ false\}$$

(Here and further 2^G designates the set of all subsets of G)

The same information can also be presented as two functions:

$Sup : S \rightarrow 2^G$	$Sup(x)$ is the supply of module x
$Dem : S \rightarrow 2^G$	$Dem(x)$ is the demand of module x

In our example, $Sup(A) = \{automobile\}$ and $Dem(A) = \{money\}$.

Now we can define three new functions:

$USup$	(Unsatisfied Supply)
$UDem$	(Unsatisfied Demand)
$ASat$	(Satisfied Demand = Satisfied Supply)

Let us add generality to already defined functions Sup and Dem in the following way:

$$Supext : 2^S \to 2^G, \quad where \quad Supext(S) = \bigcup_{x \subset S} Sup(x) \tag{1}$$

$$Demext : 2^S \to 2^G, \quad where \quad Demext(S) = \bigcup_{x \subset S} Dem(x) \tag{2}$$

Hence

$$USup(S_i) = Supext(S_i) \setminus Demext(S_i) \tag{3}$$

$$UDem(S_i) = Demext(S_i) \setminus Supext(S_i) \tag{4}$$

$$ASat(S_i) = Supext(S_i) \cap Demext(S_i) \tag{5}$$

where $S_i \subset S$

The task outlined earlier issues now in searching for such S_i that satisfy $UDem(S_i) = \emptyset$

However the method has one deficiency — it is inefficient in most cases: it takes too much time to find some result. That is why we have to refuse this simple solution and change it for more complex one.

The source of a big acceleration is in recursive definition of the basic three functions.

First, let us prove a simple theorem.

Def: $P_n = \{S_i \mid i = 1 \dots n\}$ — decomposition of set S, if $\forall i : S_i \subset S$ and $\forall i, j : S_i \cap S_j = \emptyset$ and $\bigcup_{i=1}^n S_i = S$

Theorem 1. *If P_n is decomposition of set S, then*

$$USup(S) = \bigcup_{(i,j)=(1,1)}^{(n,n)} USup(S_i) \setminus UDem(S_j) \tag{6}$$

$$UDem(S) = \bigcup_{(i,j)=(1,1)}^{(n,n)} UDem(S_i) \setminus UDem(S_j) \tag{7}$$

$$ASat(S) = \bigcup_{(i,j)=(1,1)}^{(n,n)} UDem(S_i) \cap USup(S_j) \tag{8}$$

Proof:

$$\bigcup_{(i,j)=(1,1)}^{(n,n)} (USup(S_i) \setminus UDem(S_j)) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{n} (USup(S_i) \setminus UDem(S_j)) =$$

$$\bigcup_{i=1}^{n} \left(USup(S_i) \setminus \bigcup_{j=1}^{n} UDem(S_j) \right) = \bigcup_{i=1}^{n} \left(USup(S_i) \setminus UDem\left(\bigcup_{j=1}^{n} S_j \right) \right) =$$

$$\bigcup_{i=1}^{n} (USup(S_i) \setminus UDem(S)) = USup\left(\bigcup_{i=1}^{n} S_i \right) \setminus UDem(S) =$$

$$USup(S) \setminus UDem(S)$$

It follows from (3) and (4) that $USup(S) \cap UDem(S) = \emptyset$. Therefore $USup(S) \setminus UDem(S) = USup(S)$. Thus the theorem is proven for $USup(S)$. The two other statements can be proven by analogy.

In particular for any x which belongs to S_i

$$\begin{aligned}
USup(S_i) &= (USup(S_i \setminus \{x\}) \setminus Dem(x)) \cup (Sup(x) \setminus UDem(S_i \setminus \{x\})) && (9) \\
UDem(S_i) &= (UDem(S_i \setminus \{x\}) \setminus Sup(x)) \cup (Dem(x) \setminus USup(S_i \setminus \{x\})) && (10) \\
ASat(S_i) &= (USup(S_i \setminus \{x\}) \cap Dem(x)) \cup (Sup(x) \cap UDem(S_i \setminus \{x\})) && (11)
\end{aligned}$$

These expressions being more complex can be evaluated quicker than the previous (6), (7), and (8). The reason is that on every stage we use data from the previous stage $USup(S_i \setminus \{x\})$, $UDem(S_i \setminus \{x\})$ instead of calculating them from the beginning.

As it was noticed, the task is to search such S_i, that $UDem(S_i) = \emptyset$. Let us use a search procedure known as generate-and-test method [1]: generating $S_i \subset 2^S$ we will check for $UDem(S_i) = \emptyset$ via expression (10). There exist several ways to generate variants. Let us choose the optimal.

The simplest method of checking all possible subsets of S takes as more as $2^{\#S}$ tests and that is why the search time rises exponentially with $\#(S)$. So when working with 1000 offers, this method takes 2^{1000} tests.

The solution of this problem (and not the only but even three solutions) issues from expression (11) for satisfied demand. As observed it consists of the two parts. It can be presented as following:

$$\begin{aligned}
ASat(S_i) &= ASDem(S_i) \cup ASSup(S_i) && (12) \\
ASDem(S_i) &= USup(S_i \setminus \{x\}) \cap Dem(x) && (13) \\
ASSup(S_i) &= Sup(x) \cap UDem(S_i \setminus \{x\}) && (14)
\end{aligned}$$

Expression (13) shows the part of satisfied demand issued from supply of element x, and expression (14) — from demand of element x.

3 Methods

All these methods deal with expansion. We call expansion the process of appending new elements to a structure. This process includes the backtracking if the system built does not meet our standards.

3.1 Demand expansion

We start with the target element and try to expand the system taking into account only those elements that can satisfy our demand (i.e., the demand of the system already built). Mathematically $ASDem(S_i) \neq \emptyset$. The method is based on expression (13).

This method is efficient in tasks of construction of educational programs. Such tasks characterized by the minimum number of items in supply (often the only variant) for a particular demand. We would call this a producer market. The structures synthesized may often be tree-like.

3.2 Supply expansion

As for the case above but here we take into consideration only those elements that can help us to get rid of things we have for sale. Mathematically $ASSup(S_i) \neq \emptyset$. The method is based on expression (14).

This method is effective, when the supply dominates over the demand. In other words it is a consumer market. The structures synthesized may be tree-like too with the demand as the root.

3.3 Adaptive methods

The two methods already defined are efficient only for homogeneous areas of application. When the number of producers of *every* item of goods is one or two and this situation is stable, we surely use demand expansion. At the other end, in the case of a consumer market, we use supply expansion definitely. In real life the situation is more variable and often it is not strictly defined. If we apply the wrong method we will not get the result in time. That is why it is necessary to have an algorithm to choose appropriate methods based on the current situation. This algorithm can be constructed in several ways. At first we divide them into static and dynamic [1].

A static algorithm fixes the initial situation and chooses the appropriate method for this situation. The method chosen will be used throughout the synthesis assuming that the situation remains the same. This assumption is true when the situation is near homogeneous. The choice decision is made once at the beginning of the synthesis.

Dynamic algorithms analyze the situation at every step of expansion. They allow to react adequately when the situation changes. However the analysis time overhead is long. The overhead problem of dynamic algorithms is covered in [1]. Dynamic algorithm can be realized in several ways. Two are suggested here.

3.3.1 Current State Algorithm (CSA)

The basis of this algorithm is a hypothesis that a smaller number of conditions are easier to satisfy. It is not a fact in general. The hypothesis is true in many cases. It is called heuristic [2]. Such heuristics help much in reducing search time.

This is a simple dynamically adaptive algorithm. It compares the lengths of demand and supply vectors of an already constructed system at every step of expansion and makes demand expansion step (if the demand vector is shorter) or supply expansion step (otherwise).

3.3.2 Step Forward Algorithm (SFA)

As it was noted our hypothesis is false. Actually the conditions are not equal by the time of their evaluation and check. We will call it the condition difficulty. This algorithm tries to estimate the relative difficulty of the conditions to reduce its work: do what is easy and postpones what is difficult.

The algorithm is more complex than the previous one. It minimizes the search space for every step via predicate findall. Unlike the previous formula it estimates, not the lengths of demand and supply vectors, but the sizes of satisfying complementary vectors.

4 Conclusions

The realization of this approach has allowed us to obtain the following results:

1. The two basic principles and algorithms of structural synthesis were made.

2. These algorithms were realized as a dialog environment with Prolog [3].

The program supports the database of commercial offers to be entered, stored and changed. The structural synthesis of possible exchanges can be made with a method chosen by the user. The program displays already generated variants allowing us to analyze and explore them without interruption of the synthesis process.

3. Extensive tests were made to estimate the performance of the basic and adaptive methods.

   The tests have shown that the methods work. Areas of preference for each method were discovered. Method of supply expansion is preferable when applying to the field of industrial commodity exchange. However the same method is inappropriate for educational planning. This fact is connected with the intangibility of information and the infinite possibility of its duplication. For this purpose it is more convenient to use demand expansion. Also this method is applicable when dealing with consumer or monopolistic segments of market. To deal with commodity exchange in general we have to use one of the adaptive algorithms, which can change strategy depending on the situation.

5 Appendix

The algorithm of demand expansion written in Prolog:

```
chngs(Dem,Off,Sells,Trans):-
     write_offer(Off),write_demand(Dem),write_lpr(Sells),
     chng(A,DmA,OfA), not(memb(A,Sells)),
     ins(A,Sells,Sells1), not(pex(_,Sells1,_)),
     trfrm(A,DmA,DemA),trfrm(A,OfA,OffA),
     sat0(Dem,OffA,Sat_Dem,Uns_Dem,Rest_OffA,Trans_in),
     not(eq(Sat_Dem,[])),
     sat0(DemA,Off,_,Uns_DemA,Rest_Off,Trans_out),
     append(Uns_Dem,Uns_DemA,Dem1),
     append(Rest_OffA,Rest_Off,Off1),
     append(Trans,Trans_in,Transi),
     append(Transi,Trans_out,Trans1),
     chngs(Dem1,Off1,Sells1,Trans1).
```

The algorithm of supply expansion written in Prolog:

```
chngs(Dem,Off,Sells,Trans):-
     write_offer(Off),write_demand(Dem),write_lpr(Sells),
     chng(A,DmA,OfA), not(memb(A,Sells)),
     ins(A,Sells,Sells1), not(pex(_,Sells1,_)),
     trfrm(A,DmA,DemA),trfrm(A,OfA,OffA),
     sat0(DemA,Off,Sat_Off,Uns_DemA,Rest_Off,Trans_out),
     not(eq(Sat_Off,[])),
     sat0(Dem,OffA,_,Uns_Dem,Rest_OffA,Trans_in),
     append(Uns_Dem,Uns_DemA,Dem1),
     append(Rest_OffA,Rest_Off,Off1),
     append(Trans,Trans_in,Transi),
     append(Transi,Trans_out,Trans1),
     chngs(Dem1,Off1,Sells1,Trans1).
```

References

[1] Dmitry Bertsekas and Robert Gallager. *Data Networks.* Prentice-Hall, Inc, 1987.

[2] David W. Rolston. *Principles of artificial intelligence and expert systems development.* McGraw-Hill, Inc, 1988.

[3] Leon Sterling and Echud Shapiro. *The art of Prolog. Advanced Programming Techniques.* The MIT Press, 1986.