Doktorandský den '00

Ústav informatiky
Akademie věd České republiky

Praha, 22. listopad 2000

Obsah

Doktorandský den ÚI AV 2000 – program

— 22. listopad 1999 —

Časový rozvrh přednášek		
Stanislav Višnovský	Exceptions in Behavior Protocols	$8^{00} - 8^{35}$
Radek Pospíšil	Extensions of the EJB component model	$8^{35} - 9^{10}$
	přestávka 10 minut	
David Coufal	Off-line structure learning of Wang neuro-fuzzy system	$9^{20} - 9^{55}$
Zuzana Haniková	Teorie množin ve vícehodnotové logice	$9^{55} - 10^{30}$
	přestávka 10 minut	
Arnošt Štědrý	Towards feasible parallel learning algorithm based on Kolmogorov theorem	$10^{40} - 11^{15}$
	přestávka na oběd	
Zuzana Petrová	Soft Computing Agents and Bang2 - I.	$13^{00} - 13^{35}$
Pavel Krušina	Soft Computing Agents and Bang2 - II.	$13^{35} - 14^{10}$
	přestávka 10 minut	
Přemysl Žák	Cesty a úskalí imunitní sítě buněk	$14^{20} - 14^{55}$
Petr Hanzlíček	Information theoretical approach to support medical decision making using electronic patient records	$14^{55} - 15^{30}$
	přestávka 10 minut	
Petr Tichý	Vztah Lanczosovy metody k ostatním Krylovovským metodám	$15^{40} - 16^{15}$
Martin Beran	Algorithms for Decomposable BSP Computers	$16^{15} - 16^{50}$

Exceptions in Behavior Protocols

doktorand:
STANISLAV VISNOVSKY
Faculty of Mathamatics and Physics, Charles Univerity, Prague
visnovsky@nenya.ms.mff.cuni.cz

školitel:
FRANTISEK PLASIL
Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague
plasil@nenya.ms.mff.cuni.cz

obor studia:
I-2 Software systems

Abstrakt

Behavior protocols are a formal notation for describing a communication among software components. Their regular-like syntax makes them easy to comprehend while preserving expressive power required for this kind of description. However, the state-of-the-art behavior protocols are not suitable for employing in real-world applications. One of the reasons is the lack of support to make a description of the common communication patterns more convenient. In this paper, we enhance the behavior protocols by definition of a special operator allowing to describe the exception handling and by modification of already existing operators to incorporate the notion of exceptions. The proposed operator is inspired the traditional try-catch semantics of programming languages dealing with exceptions. This way it clearly separates the exception handling from the correct communication. Furthermore, the enhancements retain the decideability properties of original behavior protocols.

1. Introduction

A lot of effort is being but into formal or semi-formal description of their semantics to help the development of the software systems based on component concept. The target is to improve the specification by reducing the number of errors and by detecting errors earlier in the process.

One of the approaches are *behavior protocols* introduced in Plasil et al [1]. Their purpose is to describe a communication among software components by means of events. However, the state-of-the-art behavior protocols are not suitable for employing to real-world applications. One of the problems is the lack of support to make a description of the common communication patterns more convenient.

The exceptions represent a typical kind of communication in distributed software systems, e.g. CORBA [4]. Thus, it is natural to require their support in any notation used for describing this kind of systems. One of the first approaches was the notation for exception handling in descriptions of event-based communication in Sousa and Garlan [3]. A similar notation was introduced on ad hoc basis into behavior protocols by Pospisil [2]. However, the ad hoc approach is enough. Therefore, we define formally the necessary operators and present the necessary modifications of the current behavior protocols specification.

To target the goal, the paper is structured as follows. In Section 2 we present behavior protocols. The proposed solution for exception handling is in Section 3 and an example of the usage is in Section 4. Section 5 evaluates the newly defined notation and concludes the paper.

2. Behavior Protocols

Behavior protocol (*protocol* for short) is a regular-like expression which syntactically generates traces. A trace is a sequence of action tokens, each of them representing exactly one event-based action in a system, e.g. the action token $!d.Insert \downarrow$ representing emmiting (prefixed by !, ? is used for absorbing) the response (suffixed by $\downarrow$, $\uparrow$ is used for a request) of the $d.Insert$ invocation. The simplest behavior protocol is an action token from ACTs (the set of all action tokens in a system) or the NULL symbol for empty trace. A protocol can be constructed in similar way as a regular expression and can use the operators + (alternative), ; (sequence), * (repetition), | (parallel), / (restriction), and some others. Furthermore, there is defined the set of abbreviations for describing procedure calls, e.g. $!d.Insert$ being equivalent to $!d.Insert \uparrow ; !d.Insert \downarrow$. For definitions of the operators, see technical report [1].

To demonstrate behavior protocols, let us consider the following behavior protocol:

```
!da.Open ;
(
?d.Insert {! tr.Begin ; !da.Insert ; !lg.LogEvent ; (!tr.Commit + !tr.Abort ) } +
?d.Delete {! tr.Begin ; !da.Delete ; !lg.LogEvent ; (!tr.Commit + !tr.Abort ) } +
?d.Query { !da.Query }
)* ;
!da.Close
```

This example presents a protocol for describing a communication of a database front-end component. This front-end changes the session-oriented communication of back-end database into sessionless and provides a logging facility of the database modifications. The methods invocations prefixed by ! represents invoking of methods while ? represents receiving a method call from the environment, i.e. other components. The curly brackets, e.g. $?d.Insert\{...\}$, denote the protocol describing communication which takes place between a method d invocation and the emmiting reply to the invocation. Inside every $d.Insert$ invocation, any number of $da.Insert$ calls can be executed, and after each of these calls is finished, the modification is logged by invoking $lg.LogEvent$.

The protocol generates the traces like $!da.Open\uparrow, ?da.Open \downarrow, ?d.Query \uparrow, da.Query \uparrow, ?da.Query \downarrow, !d.Query \downarrow, !da.Close \uparrow, ?da.Close \downarrow$. The language is infinite because of the $*$ operator. However, every trace starts with a pair of action tokens representing an invocation of $!da.Open$, then it follows the handling of the database operations and it is finished by a pair of action tokens representing an invocation of $!da.Close$.

3. Specifying exception handling

Nowadays, software component are specified in abstact definition languages, e.g. CORBA IDL [4]. These include typically define two basic paradigmas for returning of the method result. First of them is a return value/return method arguments. Second option is, in case of unexpected situations and errors, raising of exceptions. The exception handling, i.e. the way program deals with exceptions, is typically based on notion of try and catch blocks as follows:

```
try {
  ...
```

```
  operations can raise exceptions
  ...
}
catch( exceptions to be catched ) {
  ...
  handling of catched exceptions
  ...
}
```

Any exception raised in try block stops the execution of the block and the program continues its execution at the beginning of the catch block, which catches the raised exception. If there is no such catch block, the exception is propagated through the call stack upwards. The catch block presents the exception handling.

The behavior protocols do not provide any support for the exception-based communication. However, even with current specification of behavior protocols, it is possible to express exceptions as follows: Let $!a; !b; c!$ be a protocol specifying the sequence of invocations of a,b and c methods. If a raises an exception $e\hat{}$ (tilde denotes a general event), we could modify the protocol to take the form $!a; (?e\hat{}+!b; !c)$. However, if b can also raise e exception, following the previous pattern, we would write $!a; (?e\hat{}+!b; (?e\hat{}+!c))$. It is obvious, that this way of specification leads quickly to unreadable specifications and there is no specification of the exception handling itself. Therefore, we define the following operator for specification of exceptions:

Definition: Let A, B be behavior protocols. The *exception handling B in A* (denoted $A \triangle B$) is the language defined as follows:

$$L(A \triangle B) = \{\beta\hat{} < x > \hat{}\delta; < x > \hat{}\delta \in L(B) \wedge \beta\hat{} < x > \hat{}\gamma \in L(A) for\ min\beta\}$$
$$\cup \{< y > \hat{}\beta; < y > \hat{}\beta \in L(A) \wedge \neg\exists\delta < y > \hat{}\delta \in L(B)\}$$

Informally, the left operand of $\triangle$ represents a try block and the right operand represents a catch block. To distinguish exceptions as being an event (not a RPC request nor a RPC response), we use the general event notion of behavior protocol denoted by tilde (e.g. e~). Using the definition, the example above could be rewritten as $(!a; !b; !c) \triangle ?e$.

Furthermore, to easily use the exceptions in behavior protocols, we need to change the abbreviations of the behavior protocols defined as follows:

$$!m \quad \sim \quad !m\uparrow; (!m\downarrow + !e_1\tilde{} + \ldots !e_n\tilde{})$$

where $e_1, \ldots, e_n$ are all the exceptions specified as being throwed by the m method in its definition.

4. Example

The following example presents an improved part of a part of the protocol from Section 2 specifying an invocation of *d.Insert* by adding the exception handling.

```
?d.Insert { (!tr.Begin ; !da.Insert ; !lg.LogEvent ; !tr.Commit ) }
     △ ( ?da.DuplicateKeyException~ ; !lg.LogEvent ; !tr.Abort ;
          !d.DuplicateKeyException~
       + ?tr.RollbackOnlyException~ ; !lg.LogEvent ; !d.TransactionRolledBack~
       )
```

In this protocol, there are two exceptions being caught when handling incoming invocation of *d.Insert*. First, we are catching *da.DuplicateKeyException* being returned by a *da.Insert* invocation. In this case, we log the exception by *lg.LogEvent* and then we rollback the transaction associated with the insert operation. At the end of the sequence, we propagate the exception via sending general event $d.DuplicateKeyException$. For $tr.RollbackOnlyException$ being caugth from invocation of $tr.Commit$, we do the logging by $lg.LogEvent$ and then propagating the exception by throwing a different exception – $d.TransactionRolledBack$. As the transaction is to be rollbacked anyway, we do not specify it explicitly in the protocol.

5. Evaluation

In this paper, we proposed an enhancement of behavior protocols for specification of exception handling. We addressed the problem of the easy to read notation by defining a new $\triangle$ operator and by modifying existing behavior protocol abbreviations. The operator allows easily to describe the try-catch semantics of exceptions supported in programming languages.

An important issue of any enhancement of behavior protocols is decideability of specification testing. As the behavior protocols [1] preserve the regularity of generated languages, the new operator should also. In fact, this it true, since if we have finite state machines for protocols A and B, we could easily build a finite state machine for $A \triangle B$ as follows. If a node in M_A contains transition edge marked by the action token representing an exception handled by B, it is replaced by a transition edge from the node to the target node of the transition in M_B. This edge represents the start of exception handling. Therefore, all the results in [1] hold for the behavior protocols with the $\triangle$ operator.

One of the open issues presented in [1] is the so-called internal/external choice problem. The key in a proposed solution is the possibility to transfer information about choice being made by component internally to the external client. As mentioned in Section 3, exceptions present a way to pass the information about errors or unexpected situations to the client. Therefore, the $\triangle$ operator allows to solve the internal/external choice problem.

However, there are some open issues with the exception handling in behavior protocols. One of the most importat is that the $\triangle$ operator does not ensure the protocol to return at least some response by means of the $\downarrow$ suffix or an exception raising. Thus, it is possible to write a protocol, which specifies that there will be no reply to the request issued. Although this also holds for behavior protocols as defined in [1], there are defined sound abbreviations to guide the user to write semantically correct protocols. For exception handling, there should be also proposed reasonable abbreviations for ensuring that all the time there is at least some reply to a request. This should also incorporate an "automatic" propagation of not caught exceptions.

References

[1] F. Plasil, S. Visnovsky, M. Besta, "Behavior Protocols", *TR 7/2000, Charles University*, Prague, 2000.

[2] R. Pospisil, F. Plasil, "Describing the functionality of EJB using behavior protocols", *presented at Doctorand's day, Computer Science Department*, Prague, 1999.

[3] Joao Pedro Sousa, David Garlan, "Formal Modeling of the Enterprise JavaBeansTM Component Integration Framework," *World Congress on Formal Methods*, 1999.

[4] CORBA Specification 2.3, http://www.omg.org

Extensions of the EJB component model

doktorand:
MGR. RADEK POSPÍŠIL
KSI MFF UK, Malostranské náměstí 25, 110 00 Prague 1
rpos@cs.cas.cz,pospisil@nenya.ms.mff.cuni.cz

školitel:
PROF. ING. FRANTIŠEK PLÁŠIL, CSC.
KSI MFF UK, Malostranské náměstí 25, 110 00 Prague 1
plasil@cs.cas.cz,plasil@nenya.ms.mff.cuni.cz

obor studia:
I2 - Software Systems

Abstrakt

Contemporary component model development is getting more and more important in an industrial world. But there are many differences between industrial view and academical view on component model. Large companies implement theirs component models directly and pushes users to use as-is. On the other hand academic world is spending long time on development and refinement of theirs component models, and rarely they implement it and with pure support and development environment. The goal of this paper is to reuse ideas gathered in component system on the Enterprise JavaBeans component architecture developed by Sun Microsystem.

1. Introduction

Contemporary component systems are getting more and more important in an industrial world. When Microsoft's COM/DCOM [23] component system was introduced, Sun Microsystems answered with Enterprise JavaBeans [7] component system. In academical world there are also research groups developing component systems ([3] [18] [19]). But these component systems are not successful in the real world due to lack of implementation and lack of support from companies. On the other hand, comparing industrial and academical component models, the academical models have better design, but the industrial have better support and implementation.

The paper presents the Enterprise JavaBeans component model, identifies weaknesses of the model and presents the EOA - Enterprise Object Adaptor. The EOA is used to create generic component and architectural model of the EJB.

2. EJB Component model

In the late of 90's, the Sun Microsystems company starts to work on several Java component models suitable for large applications. In 1997 the first widely acceptable component model, Enterprise JavaBeans 1.0 [7], was presented. One year later, EJB 1.1 [8] was presented. This new standard was a refined version of EJB 1.0. Latest version of EJB 2.0 public draft 2 [9], released in 2000, is even more refined and enhanced than EJB 1.1.

Today, the EJB standard is not strictly influenced by the Sun Microsystems' only, but it is controlled by Java Community Process (JSR-000019 [27]). Thus companies (members of the community) can control, add changes to the standard. On the other hand, the standard is a property of Sun Microsystems and a company willing to implement it, has to buy a license.

2.1. EJB component

The EJB component (in the EJB terminology "JavaBean") specification is tightly bounded to the Java environment. Component has two interfaces - business interfaces, describe component's functionality, and home interface with methods controlling instance's life-cycle of the component's instances. All these interfaces are java interfaces. Component definition has to contain one more java class with business logic and code controlling its life-cycle. The last part of component definition is a deployment descriptor - an XML file containing nonfunctional properties and definition of the component's environment. The nonfunctional properties allow to modify usage of three EJB services - transaction service, persistence service, and security service. Modification of these properties can change the component's behavior very much without its rewriting. Component's environment is composed of: environmental entries (named scalar values), EJB references (named references to EJB home interfaces), resource connection factories (named references to data sources; e.g., databases) and resource environment references (named references to generic objects; e.g., JMS destination).

There are four EJB component modes - stateful session, stateless session, entity and message-driven. Each type is specially designed for specific usage: stateful-session (interface based, featuring internal state, nonpersistent, with notification of transaction state), stateless-session (interface based, without internal state, nonpersistent), entity (interface based, persistent, with full transaction behavior) and message-driven (event-based, without internal state, nonpersistent).

From the client's point of view, an EJB component is represented by two remote java classes - one is the home remote object used to create/remove the component's instances and remote business java interface for each component instance.

2.2. EJB application architecture

An EJB application is a composition of EJB components (not instances) bounded together via environmental entries. The application is instantiated by a client. The client has to create (e.g., stateful session, entity, ...) or reincarnate (only entity) component instance(s) to work with. Consequently, all other component instances are created indirectly by running component instances. The EJB application description is done via the EJB deployment descriptor. The granularity of the EJB application description is EJB components only.

3. Weak points of the EJB component model

Comparing EJB to CORBA component model (CCM) shows several pits in the design process of EJB components. There is a lot of out-component functionality - persistence, life-cycle, transactions, security, but the real component model is missing. Also the process of deployment is weak (e.g., there is no possibility to start an application indirectly - there has to be a client, which builds up the application from accessible components).

3.1. Component model

The EJB component model is easy to learn. From the implementation point of view, a component is a java class, implementing the methods required by EJB specification, and providing an interface with the component's business methods. The mode of the component type is one of the EJB component modes given by the EJB specification (see Section 2.1). To deploy the component, a user has to provide additional information to describe transactional and security nonfunctional properties given by the application. So far so good. But there are some weak points, which limits EJB usability. The first weakness is the impossibility to add user defined component modes or at least to modify existing component modes (e.g., to allow sharing of the instance of a stateful session bean). The second weakness coheres with the previous one: It is impossible to change the transactional and security services behavior of a component modes. Of course, there are transactional attributes, but there is no possibility neither to add your own transactional "attribute", nor to add your own transactional manager (similar to security service). The third weakness is related to architecture model - a component looks like a "parametrized distributed object" but not as a component composed of other components (For example, imagine a bank component - it contains teller components and a datastore component; in the current EJB specification, a bank component is a complete application - all components are accessible to the user, they can create instances of a teller component).

To solve these weaknesses we propose the possibility to specify user-defined component modes. To solve the last weakness, a component model has to include following features: nested components, multiple-interfaces (to allow different views on a component or to delegate interfaces of internal components). The corresponding modifications to the EJB component model are proposed in this paper described in Section 4.1.

3.2. Architecture model

The third weakness (lack of nested components) mentioned in the Section 3.1 is also related to the architecture model of the EJB. In this respect, an additional problem is the impossibility to declare a reference to a component instance in the deployment descriptor (it is similar to required interface concept used in SOFA [13]) - it is implied by the original assumption of not allowing for component nesting (e.g., in the bank example, the tellers cannot share the same datastore). Another weakness is the number of interfaces - EJB component provides component one interface only. Thus, it is not possible to make the interface of a contained component visible from its parent component. Any change to the architecture model has an impact on the current view of the transaction, persistence and security usage. The issues triggered by the modifications proposed above include: where and how should be transaction attributes employed; what will be the persistence state of an instance?

4. Extensions of EJB component and architecture models

The first issue addressed in this section is the limited number of component modes. Via new component modes, it is possible to specify a nested component model, allowing to create components composed of already existing components.

4.1. Proposed EJB component model

The current EJB component model includes only four modes of components limiting thus the area of EJB applications.

To provide more component modes, the EJB component model has to be modified - for example a dynamic component type has to be defined, the limit of the number of component interfaces has to be lifted, and the behavior of services has to be described. Modifications are described in following sections.

4.1.1 New component modes: There are a number of ways how to describe the behavior of a system - fully independent on the implementation (e.g., plain-english, CSP [10], behavioral protocols [11], finite state machines [13]), parametrization of the implementation of a system and source code. In this section we discuss possible ways of component type description.

Semantic description of component mode: As it is said, the best description is the source code. But no one gives the source-code of a commercial software for free. The most common way to describe semantics of a component type is plain-english approach, i.e., the semantics is written in documentation (e.g., there is a book covering Excel component semantics as defined by Microsoft), it consists of a written specification, state diagrams, etc. The EJB specification uses the same means to describe component types. The drawbacks of the method include: (1) it is very large (hundreds of pages), (2) informal, and (3) it is not possible to generate the source code. To solve the weaknesses of plain-english specification, declarative languages were proposed. Using such a language, it should be easy to write down a specification and, reversely, to convert the written specification into understandable language to human (plain-english) or to computer (source- code). Very popular languages are CSP, regular-like languages (behavior protocols), different notions of description of semantics, etc. But this approaches are hard to use. Descriptions tends to be large and hard to understand to a human, but it can be in principle automatically converted into source code. A problem is, that hand-written specification needs to be both written and verified by a human.

Parametrized component mode: One of the approaches is to allow modifications of component types using parametrization. In this way, the component model has to offer reasonable set of properties to change. Current EJB specification support parametrization of component types (e.g., entity beans can be reentrant), but this set of parameters cannot be modified.

4.1.2 Enterprise Object Adaptor: As discussed in the Section 4.1.1, a more suitable approach to changing behavior of the component model is adding a set of properties to the current component types and to allow to add user-written code in order to control the life cycle of a component type, and to control the security, transactions and persistency services (also Abstract Persistence Scheme as defined in EJB2.0). Such a user-written code will be called Enterprise Object Adaptor (EOA). Thus any component modes will have its own implementation of EOA (including the contemporary entity, session, and message- driven components modes). The idea of EOA is inspired by CORBA's POA (portable object adaptor) where servants are handled very similar as bean instances. In principle EOA includes the following: run-time support controls life-cycle of an instance. It is notified by the events generated by an EJB container (e.g., passivation, activation) or a services (e.g., change to a transaction state). All client events (e.g., method invocation) are supervised by this code and necessary actions are taken (e.g., transaction begun, security checked, interceptor code invoked); this code can use the services provided by a server; stub/skeleton generator which creates stub and skeleton code; it is necessary, for example, to propagate a special contexts (e.g., transaction context of an application specific transaction service); deployment code which creates and configures instances of an EOA run-time support (configuration is read from the deployment descriptor). To incorporate EOA into current EJB does not imply any modification to current component modes (entity, sessions and message-driven). Although EOA allows to create a variety of component modes, it is not intended to create specifics EOA for each component mode in an application. The actual intention is to provide a few "useful"component modes to enhance the application are of the current EJB model. In the following sections, the new EOA -related features of the proposed component model are presented.

Multiple interfaces: A component compliant to the current EJB specification has only one interface. This interface can be method-call-based (it contains methods) or event-based (supported by message-driven bean). To support hierarchical components, a component model has to allow for multiple interfaces of a component. This feature is required in order to delegate interfaces of internal components or to expose the component's functionality via different views. As a consequence, the presence of multiple interfaces allows for a component to provide both a method-call-based and event-based interfaces. However allowing for multiple interfaces rises the following issues: (1) Session stateful component instance can be accessed by one client thread (i.e., it is not possible to use an component instance by other threads except for its creator thread). (2) Entity component instance can be accessed by more clients/threads; unfortunatelly, it is not clear where the synchronization-related component state is to be stored (in the component instance itself or in the underlaying database). (3) Transactional attributes and security properties of parent component's interface and sub component's interface should not collide. As an aside, multiple interfaces of a component typically include a control interface that provides a method to list all other interfaces of the component and a method to navigate to a requested interface. It is clear that the control interface is the component's home-interface.

Composed components: Composed components (also called nested components) were discussed in a number of publications on component models (e.g., SOFA [13], Wright [10]). The key features of composed components includes: a composed component has a hierarchical structure, subcomponents are not directly accessible outside the component (its provides interfaces has to be delegated via parent's provides interfaces) and they cannot call directly across hierarchy (its requires interfaces has to be subsumed via parents requires interfaces). In the same way, the EJB composed components are designed The EJB composed component type should provide these following features: hierarchical structure a component is build of components (subcomponents); it is possible to share one instance of a subcomponent through one of its interface; multiple interface used to create different views of a component and/or to delegate a subcomponent interface to the parent component interface; a component provides a set of interfaces and requires a set of interfaces; delegation of subcomponent's interface used to add subcomponent's interface to component provides interfaces; subsumetion of the component's interface used to bind a subcomponent's requires interface to the components' requires interface; traversing of component internal structure for development and special usage describe component by properties in deployment descriptor description of service usage by the component and in the component (e.g., transaction, service, persistence); references to the runtime environment (e.g., databases, JMS) and properties of the runtime environment.

All these features do not collied with the current EJB specification - they can be seamlessly included into an EJB implemention. New problems identified include: (1) what is the persistent state of the component that contain a set of entity beans and stateful session beans? (2) can composed component be passivated or not?

4.2. Architecture enhancements

Architecture enhancements are tightly bound to the enhancement to the EJB component model. The EJB specification captures description of an application's architecture as a role of the application assembler, who define properties of the components (e.g., transaction attributes, mapping to databases) and environmental the properties (references to other component's homes, datasources and values of scalar properties) related to the application. The architecture description should define these features, including those features specific for the proposed model. As the component can be composed of subcomponents, it is desirable to describe where the subcomponent's instances should be deployed. This decision can be implicitly defined by a policy or some AI-based strategy (e.g., load balancing). Such a feature requires an inter-EJB-server protocol which can transfer the code and deployment descriptor of a component, and additional server- specific information (e.g., the load on a machine). A subcomponent instance can be shared by another subcomponent instance. Thus, an instance of subcomponent has to be created. To do it, it has to be described how to create the instance (e.g., session stateless bean instance does not need any additional information, because its create() method is parameter-less, but session stateful bean instance needs typed-arguments for create() method; entity instance needs typed-arguments for the create() and findByXXX() method). A solution to it can be an XML description of types and arguments.

4.3. Added features

EOA introduce a new component and architectural model for EJB. At the same time, it is possible to use EOA for anther purpose. One of the goals for new EJB specification 2.1 is to add support for interceptors (i.e., methods called when a component method or instance's state has changed). This can be easily done via EOA, because EOA is defacto an "interceptor" manager. EOA can be used as an implementation skeleton of APS (Abstract Persistency Scheme), not currently specified at the implementation level in the EJB 2.0 specification.

5. Conclusion

This paper presented the current EJB component and architectural model, and discussing their weaknesses. A new component and architectural model of the EJB was proposed, which solve all weakness identified. A key idea is to introduce Enterprise Object Adapter (EOA), that control lifecycle and service employment of the component types. This approach is seamlessly incorporated into current EJB specification with respect to backward compatibility and functionality. This model will be tuned and implemented as a part of PEPiTA/ITEA project.

References

[1] EJB Comparison Project, Final Report, Distributed Systems Research Group, Charles University, Prague, January 2000, http://nenya.ms.mff.cuni.cz/thegroup/.

[2] R. Carg, Enterprise JavaBeans to CORBA Mapping 1.0, Sun Microsystems Inc., March 1998, ftp://ftp.javasoft.com/docs/ejb/ejb-corba.10.ps.

[3] S. Cheung, Java Transaction Service 0.95 Specification, Sun Microsystems Inc., March 1999, http://java.sun.com/products/jts/.

[4] S. Cheung, V. Matena, Java Transaction API 1.01 Specification, Sun Microsystems Inc., April 1999, http://java.sun.com/products/jta/.

[5] S. C. Dorda, J. Robert, R. Seacord, Theory and Practice of Enterprise JavaBean Portability, Carnegie Mellon University, Software Engineering Institute, Technical Note CMU/SEI-99-TN-005, June 1999.

[6] S. Krishnan: Enterprise JavaBeans to CORBA Mapping 1.1, Sun Microsystems Inc., August 1999, http://java.sun.com/products/ejb/docs.html.

[7] V. Matena, M. Hapner, Enterprise JavaBeans 1.0 Specification, Sun Microsystems Inc., March 1998, ftp://ftp.javasoft.com/docs/ejb/.

[8] V. Matena, M. Hapner, Enterprise JavaBeans 1.1 Specification, Public Release, Sun Microsystems Inc., August 1999, http://java.sun.com/products/ejb/docs.html.

[9] Enterprise JavaBeans 2.0 public release 2 Specification, Sun Microsystems Inc., August 2000, http://java.sun.com/products/ejb/docs.html.

[10] A. W. Roscoe, The theory and practice of Concurrence, Prentice Hall, 1998

[11] F. Plasil, S. Visnovsky, M. Besta; Bounding components via Protocols; TOOLS USA '99, August 1999

[12] J. van den Bos, C. Laffra, PROCOL: A Concurrent Object-Oriented Language with Protocols Delegation and Constraints, Acta Informatica, Springer-Verlag, 1991, pp. 511-538.

[13] Plasil, F., Balek, D., Janecek, R., "SOFA/DCUP Architecture for Component Trading and DynamicUpdating.", In Proceedings of ICCDS '98, Annapolis, IEEE CS, 1998, pp. 43-52.

[14] Frantisek Plasil, Stanislav Visnovský, Miloslav Besta, "Behavioral Protocols and Components", TOOLS 1999

[15] Jaroslav Gergic, "Versioning in SOFA/DCUP", Charles university, master theses 1999

[16] Allen, R. J., "A Formal Approach to Software Architecture." Ph.D. Thesis, School of Computer Science,Carnegie Mellon University, Pittsburgh, 1997

[17] Ralph Melton and David Garlan, "Architectural Unification", School of Computer Science,Carnegie Mellon University, Pittsburgh, 1997

[18] Medvidovic Nenad, "The C2 style", http://www.ics.uci.edu/pub/arch/c2.html

[19] Luckham D., "Rapide project", http://pavg.stanford.edu/rapide/rapide.html

[20] David C. Luckhama, James Vera, Sigurd Meldal, "Three concepts of System Architecture", Stanford University, 1995

[21] Object Management Group, "CORBA Component model" http://www.omg.com

[22] "Object Management Group" http://www.omg.com

[23] Microsoft "COM/DCOM" http://www.microsoft.com/com

[24] Sun Microsystems "JavaBeans" http://java.sun.com/beans

[25] Sun Microsystems "Java Messaging System" http://java.sun.com/jms

[26] G. Kiszales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loing, J. Irwin, "Aspect-Oriented Programming", Xerox Palo Alto Research Center

[27] Sun Microsystem "Java Specification Request" http://java.sun.com/aboutJava/communityprocess/jsr.html

Off-line structure learning of Wang neuro-fuzzy system

doktorand:
Ing. David Coufal
Institute of Computer Science
coufal@cs.cas.cz

školitel:
Doc. Ing. Stanislav Krejci, CSc.
University of Pardubice
krejci@upce.cz

obor studia:
technical cybernetics

Abstrakt

The paper is a rearranged part of my Ph.D. thesis which deals with employment of neuro-fuzzy systems for identification tasks. Within the paper the idea of neuro-fuzzy systems is presented together with short review of Wang fuzzy system and radial basis function (RBF) neural networks. Further it is presented how this fuzzy system and RBF neural network can be combined into neuro-fuzzy system. The main goal of the paper is to present new algorithm of off-line structure learning of this neuro-fuzzy system.

1. Introduction

According to Zadeh [1] soft computing research area should be considered as area for investigation of creative fusion of three main (possibly others) nature inspired computational models - neural networks, fuzzy systems and genetic algorithms. Apparently, the most developed domain from this point of view is a domain of combination of neural networks with fuzzy systems, especially the area of what is now called as *neuro-fuzzy systems*. According to [10] this preference is probably due to fact that neural networks and fuzzy systems - especially fuzzy controllers - became popular at the same time, at the end of the 80's. Those applying fuzzy controllers, having problems to tuning them, have admired the apparent ease with which neural networks learned theirs parameters. Therefore intensive effort was aimed on combination of these two computational models. However, at this time numerous papers dealing with combination of other two or even all three computational models are published and the interest in the soft computing area is growing.

Now let us shortly characterize fuzzy systems and neural networks respectively. Before we do it remain that both computational models perform some function $f_s : \mathcal{R}^n \to \mathcal{R}^m$. This function, actually a system representing it, is build on base of some other, partially known, master function f_m. An information about f_m is either given in a form of vague linguistic terms - the case of fuzzy system's representation of f_s - or by set of input/output examples - the case of neural network's representation of f_s. The intention behind a building a system is to represent master function f_m by function f_s as good as possible. Therefore in both cases a task of system building is actually a task of function approximation. However each approach, fuzzy or neural, has different methodology and different form of initial information.

Fuzzy systems are known as computational models whose are capable to acquire and process information given in a form of vague linguistic terms commonly used by humans. To accomplish this task they represent linguistic terms as fuzzy sets whose are combined into form of IF-THEN rules to express more complex linguistic information. Particular fuzzy sets are then given by theirs membership functions typically represented by some parametric functions from $\mathcal{R}$ to interval $[0, 1]$.

When we are setting a fuzzy system's real application we are typically encountered with a problem of particular rules and fuzzy sets specification. Standard approach is to ask an expert for rules and meaning of employed linguistic terms. However, even if it can be without problems done - an expert is at our disposal, the shapes of fuzzy sets has to be usually tuned (tuning of values of membership functions parameters) to fuzzy system would achieve desired performance because obtained representation of linguistic terms from expert is always in some degree subjective.

Manual tuning of given fuzzy system on base of trials and errors is a tedious task and it would be nice to have some tool/algorithm which enables to set required parameters on base of given experimental data. We know that especially neural networks are able to effectively solve this task. So it is natural to seek for some kind of fuzzy systems and neural networks combination which enables to utilize learning capability of neural networks when building a fuzzy system. However, resulting form of combination is required to do not destroy the main feature of fuzzy systems - linguistic interpretability in the form of IF-THEN rules.

Now let us consider neural networks. As it is well known, the main feature of neural networks is theirs learning ability. Utilizing some learning algorithm they can (theoretically in parallel fashion) acquire information given by set of examples. However, standard neural network is a black box. When a network is learned we have acquired information but this information is encoded in actual setting of network's parameters without any straightforward meaning. Therefore it would be advantageous to give some rearrangement of neural networks paradigm to retain ability of learning but to have possibility to "easily" interpret the learned information typically in form of some rules. Since by rules interpretation of information is a native feature of fuzzy systems we have again here a reason for investigation of neural networks and fuzzy systems combination.

From the above discussion it is clear that an investigation of neuro/fuzzy combination is reasonable and it should issue in a qualitatively new computational model which retains advantages of both "parent" approaches. Of course, there are several other views of this fusion presented in the literature, not only the one induced by the above discussion. To have an unified view of current approaches they can be sorted into three main groups.

- **neuro-fuzzy systems**
  Neuro-fuzzy systems are fuzzy systems realized in a neural network fashion. Neural network architecture brings into fuzzy systems learning ability known from neural networks theory, but despite of architecture, a neuro-fuzzy systems can be still considered as (certain type of) fuzzy systems. Neuro-fuzzy systems then combine main features of neural networks and fuzzy systems - ability to learn from examples and ability to incorporate and process vague information given in a form of linguistic terms.

- **fuzzy-neuro systems**
  Fuzzy-neuro systems are enhanced neural networks endowed by fuzzy weights and/or capable to compute with fuzzy inputs. More specifically, inputs, weights and outputs of a general fuzzy-neuro system are fuzzy numbers. A single neuron in such a system then performs operations based on arithmetic of fuzzy numbers [2]. For a survey about these systems with extensive references on relevant literature see [11].

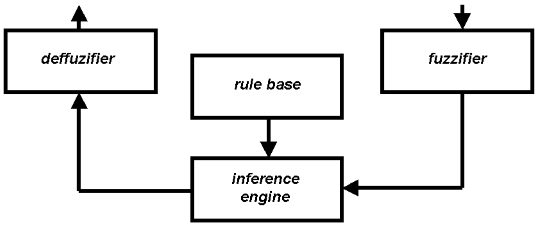
- **cooperative neuro/fuzzy systems**
  These systems can be characterized as non-homogenous combinations of neural networks and fuzzy systems to improve the performance of one of these systems. Typical example is when fuzzy system is used to operate learning rate of neural network's learning process; or when one system is used for some kind of preprocessing/postprocessing of input/output data of the other system.

The approaches of first two groups represent homogenous combinations of considered paradigms resulting into the new computational models. In opposition, approaches of the third group represent solutions with only mutual intercommunication between fuzzy system and neural network. Regarding homogenous combinations currently domination of neuro-fuzzy systems in applications can be observed. Partial reason for this fact is high computation effort required for fuzzy-neuro systems than for neuro-fuzzy ones.

The aim of my thesis is to investigate applicability of neuro-fuzzy systems for identification tasks. Therefore in the following text we will proceed only with neuro-fuzzy systems.

2. General architecture of fuzzy systems / Wang fuzzy system

It is well known that general fuzzy system consists of four building blocks *a fuzzifier*, *a rule base*, *an inference engine* and *a defuzzifier* [2]. These blocks are mutually interconnected as presented in Fig.



Obrázek 1: Architecture of general fuzzy system.

A general fuzzy system with n inputs and m outputs (system of (n, m)-type) performs some function $f_{mimo} : \mathcal{R}^n \rightarrow \mathcal{R}^m$, i.e., in control terminology, it represents MIMO (multiple input multiple output) system. It can be shown that MIMO fuzzy system can be seen as parallel connection of m MISO (multiple input single output) systems. From this reason it is sufficient to investigate fuzzy systems in MISO configuration ($(n, 1)$-type) which performs some function $f_{miso} : \mathcal{R}^n \rightarrow \mathcal{R}$.

Fuzzifier: A fuzzifier performs a process of fuzzification, which is a mapping from input space $X \subseteq \mathcal{R}^n$ into the set $\mathcal{F}(X)$ of all fuzzy sets defined on this space.

$$fuzz : X \rightarrow \ \mathcal{F}(X). \tag{1}$$

The work of a fuzzifier can be seen as transformation of crisp but inprecise (due of noise, error of measurement) input to a fuzzy set which just represents the inprecise nature of the input.

In practice, general fuzzifier is often replaced by so called *singleton fuzzifier*. In this case a crisp input $\boldsymbol{x}^* \subseteq X$ is related with corresponding *fuzzy singleton*. Hence, $fuzz$ function has the form:

$$fuzz(\boldsymbol{x}^*) = A'_{\boldsymbol{x}^*}(\boldsymbol{x}) = \begin{cases} 1 & \text{for } \boldsymbol{x} = \boldsymbol{x}^* \\ 0 & \text{for } \boldsymbol{x} \neq \boldsymbol{x}^* \end{cases} . \tag{2}$$

The employment of singleton fuzzifier represents a situation when we consider given crisp input $\boldsymbol{x}^*$ as precise enough for our purposes. Other and probably more influential reason for employing singleton fuzzifier is a simplification of fuzzy system output computation [2, 3].

Rule base (MISO): A rule base of fuzzy sytem is mathematically a fuzzy relation defined on cartesian product $X \times Y$, of fuzzy system's input and output space. It will be denoted $RB(X, Y)$, note $X \subseteq \mathcal{R}^n$, $Y \subseteq \mathcal{R}$. Canonical form of rule base is given by a set of IF-THEN rules. Single rule is also a fuzzy relation defined on universum $X \times Y$. Traditional linguistic form of a (jth) rule is

$$R_j : \textbf{IF } x_1 \ is \ A_{j1} \ \textbf{\textit{and}} \ \ldots \ \textbf{\textit{and}} \ x_n \ is \ A_{jn} \ \textbf{THEN } y \ is \ B_j, \tag{3}$$

where A_{ji}, $i = \{1, \ldots, n\}$ and B_j are fuzzy sets represented by linguistic terms. Membership function of fuzzy relation represented by rule (3) is given by formula

$$R_j(\boldsymbol{x}, y) = R_j(x_1, \ldots, x_n, y) = (A_{j1}(x_1) \star \ldots \star A_{jn}(x_n)) \triangleright B_j(y), \tag{4}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$; $\star$ denotes a t-norm which is a representation of lingvistic conection ***and***; and $\triangleright$ is a operation which will be defined later. Sometimes a shorter notation of a particular rule is used in the form

$$R_j : A_j(\boldsymbol{x}) \triangleright B_j(y) \ , \tag{5}$$

where $A_j(\boldsymbol{x})$ is so called antecedent and $B_j(y)$ so called succedent of jth rule. Antecedent then has then form

$$A_j(\boldsymbol{x}) = A_{j1}(x_1) \star \ldots \star A_{jn}(x_n). \tag{6}$$

Now there are two basic approaches how the overall rule base is combined from particular rules. That is, how fuzzy relation $RB(X, Y)$ is constructed on base of particular relations $R_j(X, Y)$. In the following, we consider the rule base consisting of m rules, i.e., $j = \{1, \ldots, m\}$.

- DC-type (disjunction of conjunction)

$$RB(\boldsymbol{x}, y) = \bigcup_{j=1}^{m} R_j(\boldsymbol{x}, y), \tag{7}$$

$$RB(\boldsymbol{x}, y) = \bigcup_{j=1}^{m} \left[(A_{j1}(x_1) \star \ldots \star A_{jn}(x_n)) \triangleright B_j(y) \right]. \tag{8}$$

  Union $\bigcup$ operation is generally given by some t-conorm, however, maximum operation is typically used. The operation $\triangleright$, in DC-type representation of rule base, is given by a t-norm, usually the same which is used in antecedents of rules. Membership function of $RB(X, Y)$ has then the final form

$$RB(\boldsymbol{x}, y) = \bigcup_{j=1}^{m} \left[A_{j1}(x_1) \star \ldots \star A_{jn}(x_n) \star B_j(y) \right]. \tag{9}$$

- CI-type (conjunction of implication)

$$R(\boldsymbol{x}, y) = \bigcap_{j=1}^{m} R_j(\boldsymbol{x}, y), \tag{10}$$

$$R(\boldsymbol{x}, y) = \bigcap_{j=1}^{m} \left[(A_{j1}(x_1) \star \ldots \star A_{jn}(x_n)) \triangleright B_j(y) \right] \tag{11}$$

  Intersection $\bigcap$ operation is generally given by some t-norm, however, minimum operation is typically used. The operation $\triangleright$, in CI-type representation of rule base, is given by a fuzzy implication. Membership function of $RB(X, Y)$ has then the final form

$$R(\boldsymbol{x}, y) = \bigcap_{j=1}^{m} \left[(A_{j1}(x_1) \star \ldots \star A_{jn}(x_n)) \rightarrow B_j(y) \right]. \tag{12}$$

Within the applications, however, the first type (DC) of construction is almost always used. The reason is that with second type (CI) there are severe computational difficulty regarding fuzzy implication.

Inference engine: An inference engine performs a mapping from cartesian product of $\mathcal{F}(X) \times \mathcal{F}(X, Y)$ to $\mathcal{F}(Y)$; $\mathcal{F}(X, Y)$ is the set of all fuzzy relations defined on $X \times Y$. That is, an inference engine makes projection of fuzzy set given by fuzzifier through fuzzy relation given by rule base into a fuzzy set defined on output space Y. Membership function of output fuzzy set is given by so-called compositional rule of inference (CRI) [2]. It has the form:

$$B'(y) = \sup_{\boldsymbol{x} \in X} \left[A'_{x^*}(\boldsymbol{x}) \star RB(\boldsymbol{x}, y) \right], \tag{13}$$

where $\star$ is generally some t-norm.

The solution of CRI formula with respect to explicit form of $B'(y)$ cannot be generally given in advance i.e., (13) cannot be further rearranged until forms of A'_{x^*} and RB are given. When fuzzy set A'_{x^*} is generated on base of singleton fuzzifier then the following simplification can be made. In the case of singleton fuzzifier the value of $A'_{x^*}(\boldsymbol{x})$ is equal to one only at point $\boldsymbol{x}^*$, otherwise it is zero. Since for each t-norm i, $i(0, a) = 0$ holds, the CRI rule for singleton fuzzifier's A'_{x^*} has the form:

$$B'(y) = RB(\boldsymbol{x}^*, y). \tag{14}$$

The possibility of this simplification induces the popularity of singleton fuzzifier for fuzzy systems design purposes.

Defuzzifier: A defuzzifier performs a process of deffuzification which is mathematically a mapping from set of all fuzzy sets defined on output space Y to crisp points of this output space

$$defuzz : \mathcal{F}(Y) \rightarrow Y. \tag{15}$$

A process of defuzzification is, in fact, reverse process to process of fuzzification performed by fuzzifier. The purpose of deffuzification is to convert fuzzy set given by inference engine $B'(y) \in \mathcal{F}(Y)$ to point $y^*_{B'} \in Y$. This task is rather complicated because we need to compress information given by fuzzy set to one point of universum the fuzzy set is defined on. That is why there are many methods of deffuzification suggested in the literature. Unfortunately, nowadays there is no unified complex approach to justify among them. In practical applications, however, the most often used methods are the ones with the less computational complexity.

At this place, with respect to need of our work, we mention only two deffuzification method. Other methods can be found anywhere [2, 3, 4]. Remain again that we consider the case of $Y \subseteq R$.

- ***Center of Area/Gravity method (COA):*** This is the most often method reffered in the literature. The idea behind this method is to transform given fuzzy set to point at which the area under the graph of membership function is divided into two equal subareas. This point is given by formula:

$$y^*_{B'} = \frac{\int y \cdot B'(y)\, dy}{\int B'(y)\, dy}. \tag{16}$$

  Formula (16) is universal, applicable to all reasonable (integrals in (16) exist) fuzzy sets $B'(y)$. However, if $B'(y)$ is a general fuzzy set, then computation of integrals in (16) is hard because it cannot be usually given in analytic form. In this case numeric computation has to be taken which is time consuming. The way of handling this problem is to take into the account the form of $B'(y)$'s construction.

- ***Centroids methods (CM, CMs)***: Formulation of CM method (sometimes called as centroid average defuzifier) results from effort for computational simplification of COA method. Consider the case of composition of $B'(y)$ from m fuzzy sets $B_1(y), \ldots, B_m(y)$ combined together by minimum. Then as approximation of (16) it can be taken formula

$$y^*_{B'} = \frac{\sum_{j=1}^{m} B_j(c_j) \cdot c_j}{\sum_{j=1}^{m} B_j(c_j)}, \tag{17}$$

  where c_j is so called centroid of fuzzy set B_j, given as

$$c_j = \frac{\int y \cdot B_j(y)\, dy}{\int B_j(y)\, dy}, \tag{18}$$

i.e., centroid c_j is given by application of COA method on fuzzy set B_j. Sometimes (and it will be also our case) is denominator in (17) omitted and deffuzification is performed in only in the form of

$$y^*_{B'} = \sum_{j=1}^{m} B_j(c_j) \cdot c_j, \tag{19}$$

which will be called as simplified centroid method (CMs).

Centroids $c_1, \ldots, c_m$ can be (possibly numerically) computed in advance, when fuzzy system's rule base is formed and need not be recomputed during fuzzy system computation. These two factors together with mathematical form of (17) or (19) substantially simplify computation of deffuzified value in opposition to situation of COA's original method formulation where integrals of (16) has to be recomputed in each step of fuzzy system's computation.

Wang fuzzy system: Currently when we gave description of basic building blocks of a general fuzzy system we can proceed to particular example of a fuzzy system. This example will be so-called Wang fuzzy system.

Wang fuzzy system (WFS) was proposed by Wang in [5, 6]. This fuzzy system is based on employment of gaussians functions for fuzzy sets representation and product t-norm for representation of *and* connective in particular rules. Wang fuzzy system is considered in MISO configuration with singleton fuzzifier. Rule base is represented as DC type.

Let's start with gaussians function which represent membership functions of fuzzy sets in WFS. Gaussian function is given by well know formula, which will be writen in the following form

$$g(x) = a \cdot exp\left[-\alpha(x-\beta)^2\right], \tag{20}$$

where, using probability terminology, β is mean value and α is usually writen in form of $\alpha = 1/(2\sigma^2)$, where sigma is variance. a is a scalling parameter, which is set equal to unity when (20) represents a fuzzy set.

In the case of product t-norm and DC type of construction or rule base we represent $\triangleright$ and $\star$ symbols as products and we have (5) in the form

$$R_j(\boldsymbol{x}, y) = A_j(\boldsymbol{x}) \cdot B_j(y), \tag{21}$$

with antecedent given as

$$A_j(\boldsymbol{x}) = A_{j1}(x_1) \cdot A_{j2}(x_2) \cdot \ldots \cdot A_{jn}(x_n). \tag{22}$$

Considering A_{ji} and B_j to be gaussians

$$A_{ji}(x_i) = exp\left[-\alpha_{ji}(x_i-\beta_{ji})^2\right], \;\; B_j(y) = exp\left[-\alpha_j(y-\beta_j)^2\right] \tag{23}$$

we have for (22)

$$A_j(\boldsymbol{x}) = \prod_i \left(exp\left[-\alpha_{ji}(x_i-\beta_{ji})^2\right]\right), \tag{24}$$

which can be rearranged according to gaussians properties into the form

$$A_j(\boldsymbol{x}) = exp\left[-\sum_{i=1}^{n} \alpha_{ji}(x_i-\beta_{ji})^2\right]. \tag{25}$$

Now when singleton fuzzifier is employed together with DC type of rule base composition we have for output fuzzy set $B'(y)$

$$B'(y) = \bigcup_{j=1}^{m} A_j(\boldsymbol{x}^*) \cdot B_j(y). \tag{26}$$

Considering simplified centroids method (CMs) and the fact that B_j's centroid is given by value of β_j and the fact that $B_j(\beta_j) = 1$, we have for overall computation of Wang fuzzy system the following formula

$$y^* = \sum_{j=1}^{m} A_j(\boldsymbol{x}^*) \cdot \beta_j, \tag{27}$$

which is employing (25)

$$WFS_{MISO}(\boldsymbol{x}^*) = \sum_{j=1}^{m} exp\left[-\sum_{i=1}^{n} \alpha_{ji}(x_i^* - \beta_{ji})^2\right] \cdot \beta_j. \tag{28}$$

3. RBF neural networks / Gaussian RBF networks

Radial basis function (RBF) neural networks are beside perceptron networks the second large group of networks used in applications. General computation of RBF network's neuron is mainly given by specification of so-called radial basis function. Radial basis function is a function which is symmetrical about a given centre in a multidimensional space. The most common way of radial basis function's definition, $rbf : \mathcal{R}^n \to \mathcal{R}$, is to give it in the form

$$rbf(\boldsymbol{x}) = f(||\boldsymbol{x} - \boldsymbol{\beta}||), \tag{29}$$

where $\boldsymbol{x} \in \mathcal{R}^n$; $\boldsymbol{\beta} \in \mathcal{R}^n$ is a center the rbf is symmetric arround; $||\cdot||$ is some norm in $\mathcal{R}^n$ and $f : \mathcal{R} \to \mathcal{R}$ is a decreasing function. Radial basis functions of form (29) then represent RBF network's neuron computation

$$rbfneuron(\boldsymbol{x}) = f(||\boldsymbol{x} - \boldsymbol{\beta}||). \tag{30}$$

To equation (30) matches common concept of general neuron computation

$$o(\boldsymbol{x}) = act(agr(\boldsymbol{x})), \tag{31}$$

we can decompose (30) to aggregation function $agr : \mathcal{R}^n \to \mathcal{R}$ and activation function $act : \mathcal{R} \to \mathcal{R}$ in the following way. The aggregation function is given by some norm in $\mathcal{R}^n$, i.e.,

$$agr(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{\beta}||, \tag{32}$$

where as standard choice Euclidean is commonly used

$$||\boldsymbol{x} - \boldsymbol{\beta}||^2 = \sum_{i=1}^{n}(x_i - \beta_i)^2. \tag{33}$$

As activation function f function of (30) is taken. As it was stated it can be any decreasing function from $\mathcal{R}$ to $\mathcal{R}$. The most prominent choice for f is the exponential function

$$act(s) = f(s) = exp(-\alpha s), \tag{34}$$

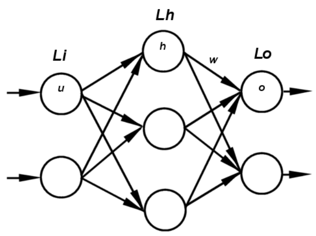
where $\alpha > 0$.

When we combine aggregation function (33) with activation function (34) we get for the whole RBF neuron's computation expression

$$rbfunit(\boldsymbol{x}) = exp(-\alpha||\boldsymbol{x} - \boldsymbol{\beta}||^2) = exp\left[-\alpha\sum_{i=1}^{n}(x_i - \beta_i)^2\right], \tag{35}$$

which forms *gaussian* type of RBF unit.

After we have single neuron's computation specified we can procced to construction of the network.



Obrázek 2: Architecture of three-layered feed-forward RBF neural network.

The most common architecture of RBF networks used in application is a three-layered feedforward network. However, networks with higher number of layers can be constructed. The notation of particular parts of RBF network is the following, see Fig.2.

In this architecture we have three layers denoted as L_I - input layer, L_H - hidden layer, L_O - output layer. Each layer consists of set of its neurons. Neurons of input layer $L_I = \{u_1, \ldots, u_n\}$ only transmit input signals into neurons of hidden layer. Neurons of hidden layer $L_H = \{h_1, \ldots, h_l\}$ are neurons of form (35). Computation of neurons of output layer $L_O = \{o_1, \ldots, o_m\}$ is usually application depend. Sometimes they are of form (35) sometimes they perform only aggregation if hidden neurons inputs

$$o_k = \sum_j w_{jk} \cdot h_j. \tag{36}$$

The interconnections of layers are feed-forward only in one direction from preceding layer to consequent layer. Interconnections endowed by weights are only the ones leading from hidden layer to output layer. Weight from neuron of hidden layer h_j to neuron of output layer o_k is denoted as $w_{jk} \in \mathcal{R}$.

4. General architecture of NFS / Wang NFS

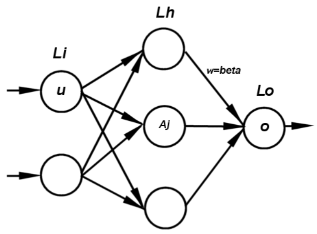
A general view of neuro-fuzzy system is a view of fuzzy system designed in neural network fashion to parameters of fuzzy system could by set according to learning algorithms known from neural networks theory. Therefore architecture of neuro-fuzzy system is determined by architecture of some type of neural network. In the area of neural networks several different architectures are recognized. However for design purposes of neuro-fuzzy systems only architectures which enable interpretability in form of fuzzy system can be employed.

It is natural to investigate combination of dominating architectures of neural networks with dominating types of fuzzy systems. From this point of view it seems as reasonable to use feed-forward networks because they propagate input in one direction (without cycling) similarly as fuzzy systems. When we considering type of a network there are two big groups - perceptron and radial basis function networks. Since fuzzy sets are typically represented by bell shaped functions, it seems to be convenient to use RBF neural networks instead of perceptrons, because RBF networks work with bell shaped functions in a native way.

The last question regarding architecture of used RBF network is its configuration in sense of number of layers used. It is natural to start with the simples case i.e., three-layered configurations. It is known from application that three-layered configuration is sufficient and in fact multilaeyred solutions can be often equivalently reformulate in three-layered fashion [10].

When we resume above paragraphs we can state that it seems to be reasonable used as underlaying network architecture three-layered RBF neural networks, with computation of particular networks utilizing gaussians and product operation, which will be case of Wang neuro-fuzzy system.

Wang neuro-fuzzy system is a Wang MISO fuzzy system represented by three-layered RBF neural network with one output neuron in the following manner, see Fig.3.



Obrázek 3: Architecture of Wang MISO neuro-fuzzy system.

First layer of the network is the standard input layer, with neurons distributing input signals into neurons of hidden layer. In fuzzy system representation they act as singleton fuzzifiers. Neurons of hidden layer then represents antecedens of particular rules of fuzzy system rule base. jth neuron computes value $A_j(\boldsymbol{x})$ of jth rule. Weights connecting neurons of hidden layers with output neurons are given (represents) values β_j, i.e., centroids of succedent of the rule. Output neurons computation is considered according to CMs method, i.e.,

$$o = \sum_j A_j(\boldsymbol{x}) \cdot \beta_j. \tag{37}$$

On base of above representation we can seen RBF neural network as fuzzy system designed in neural network fashion. Hence we can consider it as neuro-fuzzy system which has intrpreatability of fuzzy system and learning ability of neural networks. Learning of such a systems treated in next section.

5. Learning of neuro-fuzzy systems

Here we shortly mention general aspects of neuro-fuzzy systems learning. Since underlying network structure of neuro-fuzzy systems there is a significant similarity with neural networks learning. Therefore we also recognize two types of learning for neuro-fuzzy systems - *structure learning* and *parameters learning*.

Structure learning is a harder part of learning. Within this learning number of neurons/rules of neuro-fuzzy system is set together with rough assessment of parameters of membership functions of fuzzy sets. There are several approaches to this type of learning depending heavily on type of configuration - off-line or on-line. For off-line configurations there are used optimization methods known from area of nonlinear multidimensional functions' minima searching. For on-line configurations incremental learning with dynamic adding/merging/deleting of rules is used.

Parameters learning is process of tuning of neurons parameters and network's weights. It is usually based on some type of gradient descent algorithm typically on back propagation (BP) algorithm. Since aim of this paper is to present a new structure learning algorithm we will not deal with parameters learning in more details.

6. Off-line structure learning of Wang NFS

Wang neuro-fuzzy system (WFNS) was presented in section 4, there was shown that it is given by combination of Wang fuzzy system with RBF neural network. The combination is based on employment of gaussian functions for representation of fuzzy sets and for representation of activation function of network's neurons. Since gaussians employment parameters of WNFS are the following

- number of neurons in hidden layer K,

- parameters α_{ji}, β_{ji}, for each neuron h_j of hidden layer,
- weights of connections from hidden layer to output layer.

The same can be said in fuzzy system like language as

- number of rules of fuzzy system,
- parameters α_{ji}, β_{ji} of fuzzy sets building antecedent A_j of each rule,
- centroids building succedent of each rule.

Since here we are interested in learning aspects we will adopt neural network's language for description of neuro-fuzzy system.

The goal of (supervised) neuro-fuzzy system's learning, both structure learning and consequent parameters learning, is to provide a system representing a function $f_s : \mathcal{R}^n \rightarrow \mathcal{R}^m$ which match some master function $f_m : \mathcal{R}^n \rightarrow \mathcal{R}^m$ which is only partially known. The information about f_m is given by set of examples (training set) $\mathcal{T} = \{x_t, t_t\}$. Hence we are solving a task of a partially known function approximation.

From this point of view, structure learning, if it could be reasonable, should issue in such an initial setting of neuro-fuzzy system's parameters to this system represents f_m given by $\mathcal{T}$ "as best as possible". The reason is that we require for consequent parameters learning good starting setting of parameters to be theirs learning succesful.

The quatitazition of term of "as best as possible" is typically given by minimization of some metrics in $\mathcal{R}^m$ for all points in consideration. The most often choice is the Euclidean metrics in the form

$$d^2(\boldsymbol{a}, \boldsymbol{b}) = (a_1 - b_1)^2 + \ldots + (a_m - b_m)^2, \tag{38}$$

for $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}^m$.

To be our explanation clearer we will consider within the following text only the case of MISO neuro-fuzzy system i.e., system with multiple inputs ($n \geq 1$) and only one output ($m = 1$). Our considerations can be extended for general MIMO system.

The next simplification made at this time is that in equation (25) we will consider equality of α_{ji} for particular j. That is we consider $\alpha_j = \alpha_{j1} = \alpha_{j2} = \ldots = \alpha_{jn}$. With respect to this simplification and denoting β_j as c_j we can write for Wang NFS computation the equation

$$WNFS_{MISO}(\boldsymbol{x}) = G(\boldsymbol{x}) = \sum_{j=1}^{K} c_j \cdot exp\left[-\alpha_j \sum_{i=1}^{n} (x_i - \beta_{ji})^2\right]. \tag{39}$$

With repspect to Euclidean metrics a task of structure learning can be reformulated as optimization task of minimization of objective function $\mathcal{J}$

$$\mathcal{J}(K, c_1, ..., c_K, \alpha_1, ..., \alpha_K, \beta_{11}, ..., \beta_{Kn}) = \sum_s (t_s - G(\boldsymbol{x}_s))^2 \tag{40}$$

with respect to $\mathcal{J}$ parameters. Note that we are considering MISO system hence point $t_t \subseteq \mathcal{R}$. Equation (40) can be rewritten into optically more comfortable form omittig $\mathcal{J}$ parameters and explicit expressing G as

$$\mathcal{J} = \sum_s \left(t_s - \sum_{j=1}^{K} c_j \cdot exp\left[-\alpha_j \sum_{i=1}^{n} (x_i - \beta_{ji})^2\right]\right)^2. \tag{41}$$

This task can be solved by some method of multivariable optimization if number K of neurons of NFS is known in advance. Of course K can be stated a priori on base of some additional information to $\mathcal{T}$ set. But usually such information is not at our disposal. Hence K has to be set only on base of set $\mathcal{T}$, which will be discussed in the following paragraphs.

Before we state our proposal note that aim of structure learning is not necassary to obtain the mininimal solution of (41) but some approximate (tolerance to suboptimality) solution, which is further tuned by parameters learning algorithm.

Now our approach to K determination is based on incremental adding of neurons and partial optimization of objective function $\mathcal{J}_j$ of the form

$$\mathcal{J}_j(\alpha) = \sum_s \left(t_s - c_j \cdot exp\left[-\alpha \sum_{i=1}^{n} (x_i - \beta_{ji})^2 \right] \right)^2 . \tag{42}$$

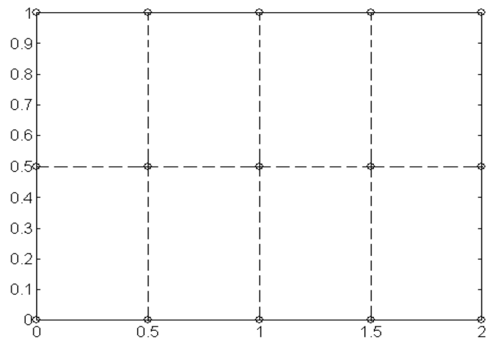
This is a function of parameters $c_j, \alpha, \boldsymbol{\beta}_j = (\beta_{j1}, \ldots, \beta_{jn})$ but consider that we know in advance values of c_j and $\boldsymbol{\beta}_j$, then the task is only in form of one-dimenzional minimization. For this task there are many numerical methods. Task of minimization of (42) with respect to α is in fact non linear least squares task, which is for gaussian not analytically solvable. To not complicate our course, consider that we are able to solve (42) with respect to α. For example see [12] for several methods.

In the above paragraph there was stated that we assume that we know values of c_j and $\boldsymbol{\beta}_j$ in (42). In the following sections we show how these values can be determined. We will show two possible approaches.

Grid specification procedure: The idea of grid specification is simple. Consider set $\mathcal{T}_x = \{\boldsymbol{x} \,|\, (\boldsymbol{x}, t) \in \mathcal{T}\}$, i.e., set of all $\boldsymbol{x} \in \mathcal{T}$. Since $\mathcal{T}_x$ is finite it can be considered as subset of cartesian product of n real intervals X_i

$$T_x \subseteq X = X_1 \times \ldots \times X_n. \tag{43}$$

Grid specification is then process of grid setting within the set X. This can be done by stating vector $\boldsymbol{r} = (r_1, \ldots, r_n)$, where r_i gives the number of lines (in geometrical representation) dividing equidistantly interval X_i to $r_i - 1$ subintervals. Instead of a precise and somewhat confusing formal definition, consider the two dimensional case, with $X_1 = [0, 2]$, $X_2 = [0, 1]$ and vector $\boldsymbol{r} = (5, 3)$ then grid within the space $X_1 \times X_2$ is displayed in Fig.4. A generalization to multidimensional case is a straightforward.



Obrázek 4: Grid specification for $\boldsymbol{r} = (5, 3)$.

Intersections of lines defined by vector $\boldsymbol{r}$ define set of *core candidates*. It will be denoted $B_{(gs)} = \{\boldsymbol{\beta}'_1, \ldots, \boldsymbol{\beta}'_{cn}\}$, where cn is cardinality of the set B. In Fig. 3. core candidates are marked by circles.

Now each core candidate $\boldsymbol{\beta}'$ will be associated with its height c' according to following rule. For given $\boldsymbol{\beta}' \in B_{(gs)}$ its height is a value of t from pair $(\boldsymbol{x}, t) \in \mathcal{T}$ such that $d^2(\boldsymbol{\beta}', \boldsymbol{x})$ is minimal with respect to all $\boldsymbol{x} \in \mathcal{T}_x$. That is, height of given $\boldsymbol{\beta}'$ is corresponding t value of point $\boldsymbol{x}$, $(\boldsymbol{x}, t) \in \mathcal{T}$, which is in the nearest distance to $\boldsymbol{\beta}'$. If there are multiple such points then one is randomly chosen.

The result of above process of heights specification is a set C of respective heights $C_{(gs)} = \{c'_1, \ldots, c'_{cn}\}$.

When we summarize grid specification procedure with respect to its inputs and outputs we can conclude the following. Inputs to procedure are set $\mathcal{T}$ of training examples and an optional parameter (vector) $\boldsymbol{r}$ defining density of grid specification. Output of procedure are two sets, set $B_{(gs)}$ of core candidates and set $C_{(gs)}$ of candidates associated heights.

Fuzzy cluster analysis: Fuzzy clustering is an alternative approach to grid specification it will also issue into stating of set of core candidates and set of associated heights.

There are several fuzzy clustering algorithms [13], but the best known algorithm is probably FCM algorithm [13] so we will consider it here. Main inputs to FCM is number of clusters denoted is as cn which need to be specified in advance.

Therefore if we specify value cn and we process FCM analysis on set $\mathcal{T}_x$, which is defined the in the same way as for grid specification procedure, we will obtain a set of cluster candidates. This set will represent set B, similarly as for grid specification $B_{(fcm)} = \{\boldsymbol{\beta}'_1, \ldots, \boldsymbol{\beta}'_{cn}\}$.

Process of setting of heights $C_{(fcm)}$ specification is the same as in the case of grid specification procedure.

Summarization of FCM analysis with respect to inputs and outputs then yields that inputs to the procedure are optional number cn of core candidates and set $\mathcal{T}$ of examples. Outputs are set of core candidates $B_{(fcm)}$ and set $C_{(fcm)}$ of associated heights.

Now we can proceed with our algorithm for number K specification and in fact with specification of other parameters of neuro-fuzzy system.

When we have at our disposal set B and associated set C from grid specification or fuzzy cluster procedure then in jth loop of algorithm the following exhaustive search is performed.

For each $\boldsymbol{\beta}'_k$ and associated height c'_k, $k = 1, \ldots, cn$ optimization task (42) is solved with respect to α. Particular solution is denoted as α_{jk} and its stored together with value $\mathcal{J}_j(\alpha_{jk})$. After this is done then the value α_{jk} with minimal $\mathcal{J}_j(\alpha_{jk})$ is chosen (we go within index k, for fixed j) and it is denoted as α^*_j. For this value corresponding $\boldsymbol{\beta}'_k$ and c'_k are specified and denoted as $\boldsymbol{\beta}^*_j$ and c^*_j. Parameters α^*_j and $\boldsymbol{\beta}^*_j$ then constitutes jth neuron in hidden layer, i.e., parameters of A_j, see (25). Value of c^*_j gives then weight of connection from jth hidden neuron to output neuron.

The last step of one loop of incremental algorithm is updating of set of examples from preceding loop $\mathcal{T}_{j-1}$. Updated set $\mathcal{T}_j$ is created as

$$\forall s, (\boldsymbol{x}_s, t_s) \in \mathcal{T}_j = (\boldsymbol{x}_s, t_s - c^*_j \cdot A_j(\boldsymbol{x}_s, \alpha^*_j, \boldsymbol{\beta}^*_j)) | (\boldsymbol{x}_s, t_s) \in \mathcal{T}_{j-1} \tag{44}$$

where $\mathcal{T}_0$ is original set of examples we start structure learning with.

The whole process of neurons/rules searching is then repeated (the next loop of algorithm $j = j+1$) with updated set $\mathcal{T}_j$ until maximal value of t within $\mathcal{T}_t = \{t \,|\, (\boldsymbol{x}, t) \in \mathcal{T}_j\}$ is less then $\epsilon \cdot max(T_0)$, where ϵ is some terminal parameter, for example $\epsilon = 0.1$. After algorithm terminates then value of j gives number of neurons K.

Algorithmic transcription of the whole algorithm for structural learning of Wang NFS in MISO configuration is as follows.

set ϵ; $T_0^{max} = max(\mathcal{T}_0)$;$j = 0$
specify sets B and C according to grid specification or fuzzy clustering
repeat
  $j = j + 1$
  for $k = 1$ **to** cn **do**
    solve minimization of (42) for c'_k, β'_k, with respect to α
    store solution α_{jk} and value $\mathcal{J}_j(\alpha_{jk})$
  end
  choose values α_j^*, c_j^*, β_j^*
  create $\mathcal{T}_j$ by updating of $\mathcal{T}_{j-1}$ according to (44)
until $max(T_j) < T_0^{max}$
number of neurons K is given as $K = j$

7. Conclusion

Within the paper fuzzy systems and RBF neural networks were very shortly reviewed. Further architecture and computation of Wang neuro-fuzzy system in MISO configuration was presented. As main result of the paper we gave new structure learning algorithm for this type of neuro-fuzzy system.

References

[1] Zadeh L.A., *What is Soft Computing?* (Editorial), Soft Computing, vol.1 (1997), no.1

[2] Klir G.J., Yuan B. Fuzzy sets and Fuzzy logic - Theory and Applications, Prentice Hall, 1995

[3] Yager R.R, Filev D.P., Essentials of Fuzzy Modeling and Control, John Wiley & Sons, 1994

[4] Driankov D., Hellendoorn H., Reinfrank M., An Introduction to Fuzzy Control, Springer-Verlag, 1993

[5] Wang L.X., Mendel J.M., *Fuzzy basis function, universal approximation, and orthogonal least-squares learning*, IEEE Trans. on Neural Networks, vol.3, no.5, pp.807-814

[6] Wang L.X., Adaptive Fuzzy systems and Control: Design and Stability Analysis, Prentice-Hall, 1994

[7] Wang L.X., A Course in Fuzzy Systems and Control, Prentice Hall, 1997

[8] Hájek P., Metamathematics of Fuzzy Logic, Kluwer Academic Publishers, 1998

[9] Rumelhart D.E., Hinton G.E., Williams R.J., Learning Internal representations by Error Propagation. In Rumelhart D.E., McClelland J.L., editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge MA, 1986, volume I, pp.318-362.

[10] Nauck D., Klawonn F., Kruse R. Foundations of Neuro-Fuzzy Systems, John Wiley & Sons, 1997

[11] Buckley J.J., Hayashi Y. *Fuzzy neural networks: A survey*, Fuzzy Sets and Systems, vol.66 (1994), no.1, pp.1-13

[12] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. Numerical Recipes in C, Second Edition, Press Syndicate of the University of Cambridge, 1992

[13] Höppner F. et al., Fuzzy Cluster Analysis, John Wiley & Sons, 1999

Teorie množin ve vícehodnotové logice

doktorand:
ZUZANA HANIKOVÁ
Ústav informatiky AV ČR, Pod Vodárenskou věží 2, 182 07 Praha 8
zuzana@cs.cas.cz

školitel:
PROF. RNDR. PETR HÁJEK, DRSC.
Ústav informatiky AV ČR, Pod Vodárenskou věží 2, 182 07 Praha 8
hajek@cs.cas.cz

obor studia:
Matematická logika

Abstrakt

Tento referát si klade za cíl informovat především účastníky konference DD2000 o vybraných způsobech rozvíjení teorie množin (jakožto teorie prvního řádu) v neklasických logikách.

Úvod

Nejprve je snad třeba upřesnit, co míním pojmem neklasické logiky; ač se totiž tento pojem vysvětluje částečně sám jakožto "logiky, které nejsou klasické", mohlo by mapříklad dojít k diskusi (a také k ní často dochází) o obsahu samotného pojmu logika. Zde je jím míněna (klasická) predikátová logika prvního řádu, tj. axiomatický systém s jistým jazykem a sémantikou, skýtající formální aparát pro matematické úvahy a představující jakýsi konsensus ohledně pravidel správného usuzování.

Neklasické logiky, jimž budeme věnovat pozornost, jsou v jistém ohledu velmi podobné logice klasické: zachovávají totiž (až na jisté drobné výjimky) jazyk logiky klasické a především chápání některých klasických pojmů, jako je formule, axiom, odvozovací pravidlo či důkaz. Od klasické logiky se odlišují sémantikou, tj. volbou algebry pravdivostních hodnot, jejíž prvky představují pravdivostní hodnoty atomických formulí a v níž dále interpretujeme logické konstanty (což jsou spojky a kvantifikátory). Povšimněme si, že tato algebra je vždy elementem "klasického" matematického světa, tj. prvkem nějakého množinového univerza generovaného klasickou Zermelo-Fraenkelovou teorií množin (ZF).

Například vícehodnotové logiky (kterými se budeme hlavně zabývat a mezi něž spadá i fuzzy logika) vycházejí z toho, že obor pravdivostních hodnot výroků je nějaká množina čítající obvykle obě pravdivostní hodnoty klasické, a ještě nějaké další. Na tuto množinu jsou obvykle kladeny poměrně restriktivní požadavky, např. bývá zpravidla svazově uspořádána, přičemž nejmenší a největší prvek odpovídají právě klasickým hodnotám 0 a 1. Pravděpodobně nejjednodušší je představovat si lineárně uspořádanou množinu nepříliš velké mohutnosti, tj. např. některou podmnožinu reálného intervalu $[0, 1]$.

V následujících odstavcích je co možná nejstručněji nastíněn formální systém BL (basic logic) a některá jeho schematická rozšíření, které obvykle používám ve vlastních úvahách. V plné šíři je tento systém rozpracován v monografii [2], z níž čerpám a z níž problematiku pochopí i ti, kteří se vícehodnotovými logikami nezabývají.

Výrokový fragment BL jsou tautologie (všech) tzv. standardních algeber, což jsou struktury na reálném jednotkovém intervalu $[0,1]$ s operacemi $*$ (spojitá t-norma, tj. binární operace, která je asociativní, komutativní, neklesající a splňuje $1 * x = x$) a $\Rightarrow$ (tzv. reziduum, které lze na reálném intervalu definovat jako $x \Rightarrow y = max\{z; z * x \leq y\}$) a (formálně) konstantou 0. Tyto operace realizují základní výrokové spojky BL, kterými jsou binární konjunkce & a implikace $\rightarrow$ a konstanta $\overline{0}$.

Jazyk predikátového počtu je obohacen o predikátové symboly a dva kvantifikátory $\forall$ a $\exists$; pravdivostní hodnota formule $\forall x\varphi$ v modelu M je infimum hodnot φ po dosazení m za proměnnou x přes všechny prvky $m \in M$, obdobně $\exists$ interpretujeme jako suprémum. Formální pravidla pro práci s kvantifikátory lze nalézt v [2].

Známá schematická rozšíření BL jsou tato: Łukasiewiczova logika, která vznikne ze systému BL přidáním výrokového schématu $\neg\neg\varphi \rightarrow \varphi$, Gödelova logika, která vznikne z BL přidáním schématu $\varphi \rightarrow \varphi \,\&\, \varphi$, a produktová logika, která vznikne z BL přidáním dvou schémat $\neg\neg\chi \rightarrow ((\varphi \,\&\, \chi \rightarrow \psi \,\&\, \chi) \rightarrow (\varphi \rightarrow \psi))$ a $\varphi \,\&\, (\varphi \rightarrow \psi) \rightarrow \overline{0}$.

Komprehenze v Łukasiewiczově logice

Prvním mně známým pokusem o (jistou) teorii množin ve vícehodnotové logice je Skolemův článek [6] z roku 1957, a pokračování [5], v nichž autor upozorňuje na možnosti využít vícehodnotových logik pro eliminaci množinových paradoxů typu Russelova.

Uvažujme formuli $\varphi(x, x_1, \ldots, x_n)$. Schéma komprehenze pro φ je trvzení

$$\forall x_1, \ldots, x_n \exists y \forall x (x \in y \equiv \varphi(x, x_1, \ldots, x_n))$$

Z tohoto tvrzení plyne v klasické logice okamžitě spor, a to i omezíme-li se na otevřené formule bez parametrů—zvolíme-li totiž za $\varphi(x)$ formuli $\neg(x \in x)$, dospějeme (po dosazení y za x) k formuli $y \in y \equiv \neg(y \in y)$. Skolem ovšem ve svém článku [7] konstruuje (v klasické logice) model schématu komprehenze pro otevřené formule s parametry, v nichž se však nevyskytují jiné spojky než konjunkce a disjunkce.

Povšimněme si nyní, že pro uvedenou konkrétní volbu φ (totiž $\neg x \in x$) přinejmenším není zjevné, že je systém sporný, nahradíme-li klasickou logiku například tříhodnotovou Łukasiewiczovou logikou—stačí uvažovat hodnotu formule $y \in y$ pro hledané y jako $1/2$. V tomto systému ovšem pochopitelně záhy opět nalezneme formule, pro něž komprehenze vede ke sporu. Článek [6] podává důkaz bezespornosti komprehenze pro otevřené formule s parametry nad nekonečněhodnotovou Łukasiewiczovou logikou. Práce [5] konstruuje spočetný model schématu komprehenze pro otevřené formule s parametry v logice s negací interpretovanou jako $1 - x$ a konjunkcí a disjunkcí interpretovanou jako minimum a maximum z obou argumentů (přičemž jiné než uvedené spojky se ve formuli φ nepřipouštějí). O tomto modelu se dále v článku prokazuje, že je extenzionální (tj. pro každé jeho dva prvky x a y, pokud pro každé u platí $\epsilon(u, x) = \epsilon(u, y)$, pak také pro každé v platí $\epsilon(x, v) = \epsilon(y, v)$, kde ϵ je funkce interpretující v modelu predikát náležení $\in$).

Booleovská univerza

Booleovské univerzum je jistým způsobem zkonstruovaná třída v rámci univerza teorie množin (např. klasické ZF). V Booleovském univerzu (nad úplnou Booleovskou algebrou v klasické ZFC) "platí" (při vhodné definici platnosti) všechny axiomy a odvozovací pravidla ZFC, tato třída je tedy "modelem" ZFC (její existence tedy prokazuje konzistenci ZFC, bohužel pouze za předpokladu konzistence opět ZFC, v níž pracujeme).

Konstrukce probíhá takto: zvolme libovolnou úplnou Booleovu algebru B. Transfinitní indukcí definujme množiny

$$V_0^B = 0$$

$$V_{\alpha+1}^B = \{f : Dom(f) \subseteq V_\alpha^B \& Rng(f) \subseteq B\}$$

pro izolovaná ordinální čísla α; pro limitní ordinální čísla λ položme

$$V_\lambda^B = \bigcup_{\alpha \in \lambda} V_\alpha^B$$

Nakonec

$$V^B = \bigcup_{\alpha \in \mathrm{On}} V_\alpha^B$$

Každé dvojici prvků $x, y \in V^B$ přiřaďme dva prvky Booleovy algebry B, které označme $||x \in y||$ a $||x = y||$ (lze chápat jako pravdivostní stupně příslušnosti x do y, případně rovnosti x a y). Tyto funkce definujme indukcí takto:

$$||x \in y|| = \bigvee_{q \in Dom(y)} (y(q) \wedge ||q = x||)$$

$$||x = y|| = \bigwedge_{q \in Dom(x)} (-x(q) \vee ||q \in y||) \wedge \bigwedge_{q \in Dom(y)} (-y(q) \wedge ||q \in x||)$$

Indukcí dle složitosti formule přiřaďme nyní každé formuli $\varphi(x_1, \ldots, x_n)$ funkci (zobrazující $(V^B)^n$ do B) takto:

$$||\neg\varphi(x_1, \ldots, x_n)|| = -||\varphi(x_1, \ldots, x_n)||$$

$$||\varphi \,\&\, \psi(x_1, \ldots, x_n)|| = ||\varphi(x_1, \ldots, x_n)|| \wedge ||\psi(x_1, \ldots, x_n)||$$

$$||\exists x \varphi(x, x_1, \ldots, x_n)|| = \bigvee_{x \in V^B} ||\varphi(x, x_1, \ldots, x_n)||$$

Případně lze prokázat i induktivní kroky pro ostatní (odvozené) logické spojky a kvantifikátor na jedné straně a příslušné Booleovské operace na straně druhé, nebo lze každou formuli nejprve vyjádřit pomocí pouze výše uvedených spojek a kvantifikátoru.

Tvrzení: Je-li φ uzavřená dokazatelná formule jazyka ZF, pak lze v ZF prokázat $||\varphi = 1||$.

Aplikace konstrukce Booleovských univerz

Princip konstrukce Booleovského univerza lze využít ke konstrukci "kanonického modelu" pro teorie množin (jakožto teorie 1. řádu) nad neklasickými logikami, popř. k prokazování relativní (vůči ZF) bezespornosti axiomatických systémů teorií množin; stačí, když v konstrukci nahradíme (úplnou) Booleovu algebru (úplnou) algebrou pravdivostních hodnot příslušné logiky (pochopitelně pak musíme přizpůsobit i pojmy typu "základní spojka" atd.). Prokazování relativní bezespornosti má pochopitelně smysl pouze v případě, kdy uvažovaný systém obsahuje v jazyce nebo v axiomatice prvky nevyskytující se v klasické ZF; jinak, jelikož neklasické logiky jsou formálně slabší než logika klasická, je relativní bezespornost zřejmá.

Tak například v článku [8] je jako výchozí algebra využita standardní algebra pro Gödelovu logiku na intervalu $[0, 1]$; axiomy, které se zde uvažují, jsou extenzionalita, $\in$-indukce, vydělení, kolekce, dvojice, sjednocení, potence, nekonečno, axiom závislého výběru a axiom existence nosiče, který říká

$$\forall x \exists z \forall u (u \in z \equiv \neg\neg(u \in x))$$

(nosič množiny x je tedy (ostrá) množina, do níž patří právě všechny množiny, které v nenulovém stupni patří do x). Posledně jmenovaný axiom umožňuje konstrukci třídy S tzv. dědičně stabilních množin, která zadává interpretaci klasické ZF v množinovém univerzu generovaném výše uvedeným axiomatickým systémem.

Tento výsledek byl původně uveřejněn v článku [3] pro logiku intuicionistickou, a převzat v článku [9], navazujícím na [8]. Práce [3] a několik jiných článků, publikovaných převážně v sedmdesátých letech (viz [1]) a rozvíjejících teorie dané Zermelo-Fraenkelovými axiomy nad intuicionistickou logikou (INT), je z hlediska vícehodnotových logik je podstatná, protože obohacením axiomatického systému (výrokové) intuicionistické logiky o jediný axiom $\varphi \to \psi \vee \psi \to \varphi$ vznikne (výroková) logika Gödelova, jedno z možných schematických rozšíření BL. Proto jsou tyto výsledky velmi zajímavé; ukazuje se například, že axiom regularity (spolu s axiomem vydělení a existence prázdné množiny) implikuje princip vyloučeného třetího ($\varphi \vee \neg\varphi$ pro libovolnou sentenci φ), který spolu s INT dává již logiku klasickou, a tento výsledek se tedy pochopitelně přenáší i do libovolného formálně silnějšího systému. Axiom regularity tedy nelze v axiomatice použít.

Konstrukci obdobnou konstrukci Booleovského univerza lze provést i uvnitř neklasického množinového univerza, ovšem až po zkonstruování příslušné algebry pravdivostních hodnot (např. nějaké standardní algebry pro fuzzy logiku apod.). V článku [9] je v rámci teorií generovaného množinového univerza V provedena konstrukce již zmíněné třídy S jakožto interpretace klasické ZF, v ní jsou pak ověřeny existence a vlastnosti klasických pojmů, jako jsou reálná čísla, a uvnitř třídy S je pak provedena konstrukce třídy $V^{[0,1]}$, analogická konstrukci Booleovského univerza, nad standardní Gödelovou algebrou $[0, 1]$. O tomto univerzu se prokazuje, že je "izomorfní" s výchozím univerzem, v němž se pracuje. K tomu je třeba mj. sestrojit uvnitř V nějakou strukturu reflektující strukturu pravdivostních hodnot použité logiky (v tomto případě Gödelovy). Touto strukturou je potenční množina množiny 1; patří do ní právě všechny podmnožiny množiny $1 = \{0\}$, a to jsou právě všechny množiny obsahující (pouze) 0 v jistém stupni (např. ve stupni 0.8).

Jinou variantou využití konstrukce Booleovského univerza lze najít v článku [4]; ač to není na první pohled patrno, pracuje se zde (jak ověřil Petr Hájek) se standardní algebrou danou Łukasiewiczovou t-normou na intervalu $[0, 1]$, obohacenou o operátor Δ (jehož sémantika vypadá takto: $\Delta(x) = 1$ pro $x = 1$, jinak $\Delta(x) = 0$). Tohoto operátoru je využito především k definici rovnosti (přesněji funkce reprezentující predikát rovnosti v kanonickém modelu), která vypadá takto:

$$||x = y|| = \Delta \bigwedge_{q \in Dom(x)} (-x(q) \vee ||q \in y||) \wedge \Delta \bigwedge_{q \in Dom(y)} (-y(q) \wedge ||q \in x||)$$

Příslušná forma axiomu extenzionality je potom $x = y \equiv \Delta(x \subseteq y) \& \Delta(y \subseteq x)$; rovnost je tedy predikát "ostrý", nabývající (ve standardním modelu) pouze hodnot 0 a 1. V takto vystavěném "modelu" lze ověřit, kromě zmíněného znění axiomu extenzionality mj. platnost axiomu prázdné množiny, dvojice, sjednocení, vydělení, kolekce a nekonečna (což je obsahem zmíněné práce).

Úkol, na němž pracuji, je přenesení této konstrukce do (poněkud obecnějšího) prostředí predikátové BL s operátorem Δ. Další axiomy, které uvažujeme a které lze konzistentně přidat, jsou axiom existence nosiče (formulace $\forall x \exists y(\mathrm{Crisp}(y) \& x \subseteq y)$, kde $\mathrm{Crisp}(y) \equiv \forall z(\Delta(z \in y) \vee \Delta(\neg z \in y))$, a axiom existence ostré potenční množiny, který praví

$$\forall x \exists z \forall u(u \in z \equiv \Delta(u \subseteq x)$$

Tento axiom v jednom konkrétním případě tvrdí, že existuje ostrá (crisp) potenční množina množiny 1 alias $\{0\}$. Jak lze snadno ověřit, tato množina opět obsahuje právě všechny množiny obsahující (pouze) 0 v jistém (pravdivostním) stupni. Na této množině lze vhodným způsobem definovat všechny operace BL-algeber a prokázat

Tvrzení: Pro libovolné axiomatické rozšíření systému BLΔ lze z axiomů tohoto rozšíření, výše uvedených množinových axiomů a axiomů rovnosti prokázat, že $P(1)$ je BL-algebra, a lze předpokládat, že platí i všechny axiomy odpovídající tomuto rozšíření (ač tuto hypotézu nemám ověřenou).

Takto sestavenou množinu axiomů (o níž z výše uvedené konstrukce plyne, že je konzistentní, a také že predikát $\in$ má ne-ostrou interpretaci) lze již rozvíjet jako formální teorii. Obvykle je potřeba nejprve vytvořit technický aparát, jako jsou ordinální čísla a indukce. Dalším obvyklým výsledkem je interpretace klasické ZF v neklasické teorii, tak jak se tomu děje v pracech [3], [9]. Je třeba ověřit, zda lze tuto konstrukci zopakovat pro BLΔ.

References

[1] R. J. Grayson, "Heyting-valued models for intuitionistic set theory", *Lect. Notes Math.*, vol. 753 (1979), pp. 402-414.

[2] P. Hájek, "Metamathematics of Fuzzy Logic", Kluwer Academic Publishers (1998).

[3] W. C. Powell, "Extending Gödel's negative interpretation to ZF", *J. Symb. Logic* 40 (1975), pp. 221-229.

[4] M. Shirahata, "Phase-valued models of linear set theory", preprint.

[5] T. Skolem, "A set theory based on a certain 3-valued logic", *Math. Scand.*, vol. 8 (1960), pp. 127-136.

[6] T. Skolem, "Bemerkungen zum Komprehensionsaxiom", *Zeitschrift f. math. Logik und Grundlagen d. Math.*, vol. 3 (1957), pp. 1-17.

[7] T. Skolem, "Studies on the axiom of comprehension", *Selected Works in Logic* (1970), pp. 703-711.

[8] G. Takeuti, S. Titani, "Intuitionistic fuzzy logic and intuitionistic fuzzy set theory", *J. Symb. Logic* 49 (1984), pp. 851-866.

[9] G. Takeuti, S. Titani, "Fuzzy logic and fuzzy set theory", *Arch. Math. Logic*, 32(1992), pp. 1-32.

[10] S. Titani, "A lattice-valued set theory", Arch. Math. Logic, 38(1999), pp. 395-421.

Towards feasible parallel learning algorithm based on Kolmogorov theorem

doktorand:
Arnošt Štědrý
ICS CAS CR
arnost@cs.cas.cz

školitel:
doc. RNDr. JiříWiedermann DrSc.
ICS CAS CR
wieder@cs.cas.cz

obor studia:
Theoretical Computer Science

Abstrakt

We present a one parallel version of learning algorithm that is based on Kolmogorov theorem concerning composition of n-dimensional continuous function from one-dimensional continuous functions. In our work we follow the approach by Sprecher who proved the most general version of the theorem as well as proposed numerical implementation details. The feasible learning algorithm is presented and its performance demonstrated, together with the discussion of further possible improvements.

1. Introduction

In 1957 Kolmogorov [5] has proven a theorem stating that any continuous function of n variables can be exactly represented by superpositions and sums of continuous functions of only one variable. The first, who came with the idea to make use of this result in the neural networks area was Hecht-Nielsen [2]. From the resemblance of the superposition formula and the form of function realized by neural network he derived that any continuous function on a n-dimensional cube can be exactly represented by a certain neural network. It was objected by Girosi and Poggio [1] that such a network requires units computing quite complicated functions, depending on the function it represents, and without any known way to train the network on a particular data.

Kůrková [3] has shown that it is possible to modify the original construction for the case of approximation of functions. Thus, one can use a perceptron network with two hidden layers containing a larger number of units with standard sigmoids to approximate any continuous function with arbitrary precision. This result was used as an argument for the universal approximation property for two-hidden-layer perceptrons. In the meantime several stronger universal approximation results has appeared, such as [6] stating that perceptrons with one hidden layer and surprisingly general activation functions are universal approximators.

In the following we review the relevant results and show how a parallel learning algorithm can be derived based on Sprecher improved version of the proof of Kolmogorov's theorem. We focus on implementation details of the algorithm, identify the problems, and show how they can be handled. Simple illustrating examples are given. Proposition of parallel versions of algoritmus finish this review.

2. Kolmogorov theorem

The original Kolmogorov result shows that every continuous function defined on n-dimensional unit cube can be represented by superpositions and sums of one-dimensional continuous functions.

Theorem 1 (Kolmogorov) *For each integer $n \geq 2$ there are $n(2n+1)$ continuous monotonically increasing functions ψ_{pq} with the following property: For every real-valued continuous function $f : \mathcal{I}^n \rightarrow \mathcal{R}$ there are continuous functions ϕ_q such that*

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \phi_q \left[\sum_{p=1}^{n} \psi_{pq}(x_p) \right]. \tag{45}$$

Further improvements by Sprecher provide a form that is more suitable for computational algorithm. Namely, the set of functions ψ_{pq} is replaced by shifts of a fixed function ψ which is moreover independent on a dimension. The overall quite complicated structure is further simplified by suitable parameterizations and making use of constants such as λ_p, β, etc.

Theorem 2 (Sprecher) *Let $\{\lambda_k\}$ be a sequence of positive integrally independent numbers. There exists a continuous monotonically increasing function $\psi : \mathcal{R}^+ \rightarrow \mathcal{R}^+$ having the following property: For every real-valued continuous function $f : \mathcal{I}^n \rightarrow \mathcal{R}$ with $n \geq 2$ there are continuous functions ϕ and a constant β such that:*

$$\xi(\mathbf{x}_q) = \sum_{p=1}^{n} \lambda_p \psi(x_p + q\beta) \tag{46}$$

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \phi_q \circ \xi(\mathbf{x}_q) \tag{47}$$

Another result important for computational realization is due to Kůrková who has shown that both inner and outer functions ψ and ϕ can be approximated by staircase-like functions with arbitrary precision. Therefore, standard perceptron networks with sigmoidal activation functions can, in principle, be used in this approach. The second theorem of hers provides the estimate of units needed for approximation w.r.t. the given precision and the modulus of continuity of the approximated function.

Theorem 3 (Kůrková) *Let $n \in \mathcal{N}$ with $n \geq 2$, $\sigma : \mathcal{R} \rightarrow \mathcal{I}$ be a sigmoidal function, $f \in \mathcal{C}(\mathcal{I}^n)$, and ε be a positive real number. Then there exists $k \in \mathcal{N}$ and functions ϕ_i, $\psi_{pi} \in \mathcal{S}(\sigma)$ such that:*

$$|f(x_1, \ldots, x_n) - \sum_{i=1}^{k} \phi_i \left(\sum_{p=1}^{n} \psi_{pi} x_p \right) | \leq \varepsilon \tag{48}$$

for every $(x_1, \ldots, x_n) \in \mathcal{I}^n$.

Theorem 4 (Kůrková) *Let $n \in \mathcal{N}$ with $n \geq 2$, $\sigma : \mathcal{R} \rightarrow \mathcal{I}$ be a sigmoidal function, $f \in \mathcal{C}(\mathcal{I}^n)$, and ε be a positive real number. Then for every $m \in \mathcal{N}$ such that $m \geq 2n+1$ and $n/(m-n)+v < \varepsilon/||f||$ and $\omega_f(1/m) < v(m-n)/(2m-3n)$ for some positive real v, f can be approximated with an accuracy ε by a perceptron type network with two hidden layers, containing $nm(m+1)$ units in the first hidden layer and $m^2(m+1)^n$ units in the second one, with an activation function σ.*

3. Algorithm proposal

Sprecher sketched an algorithm based on Theorem 2 that also takes into account Theorem 4 by Kůrková. Here we present our modified and improved version that addresses crucial computational issues.

The core of the algorithm consists of four steps: in each iteration r we first construct the mesh $\mathcal{Q}^n$ of rational points $\mathbf{d}_k$ dissecting the unit cube (cf. (50)). The functions ξ (see (54)) are defined by means of these points. The outer functions ϕ_q^j (see 52)) make use of sigmoidal steps θ defined in (53). In the end, the r-th approximation f_r of original function f is constructed according to (57) using ϕ_q^j. The terms ϕ_q^j are computed by means of previous approximation errors e_r (see (56)).

3.1. The support set $\mathcal{Q}$

Take integers $m \geq 2n$ and $\gamma \geq m+2$ where n is the input dimension. Consider a set of rational numbers $\mathcal{Q} = \left\{ d_k = \sum_{s=1}^{k} i_s \gamma^{-s}; i_s \in \{0, 1, \ldots, \gamma - 1\}, k \in \mathcal{N} \right\}$. Elements of $\mathcal{Q}$ are used as coordinates of n-dimensional mesh

$$\mathcal{Q}^n = \{\mathbf{d}_k = (d_{k1}, \ldots, d_{kn}); d_{kj} \in \mathcal{Q}, j = 1, \ldots, n\}. \tag{49}$$

Note that the number k determines the precision of the dissection.

For $q = 0, 1, \ldots, m$ we construct numbers $\mathbf{d}_k^q \in \mathcal{Q}^n$ whose coordinates are determined by the expression

$$d_{kp}^q = d_{kp} + q \sum_{s=1}^{k} \gamma^{-s} \tag{50}$$

Obviously $d_{kp}^q \in \mathcal{Q}$ for $p = 1, 2, \ldots, n$. We will make use of $\mathbf{d}_k^q$ in the definition of functions ξ.

3.2. The inner function ψ

The function $\psi : \mathcal{Q} \to \mathcal{I}$ is then defined with the help of several additional definitions. For the convenience, we follow [11] in our notation.

$$\psi(d_k) = \sum_{s=1}^{k} \tilde{i_s} 2^{-m_s} \gamma^{-\rho(s-m_s)}, \tag{51}$$

$$\rho(z) = \frac{n^z - 1}{n - 1},$$

$$m_s = \langle i_s \rangle \left(1 + \sum_{l=1}^{s-1} [i_l] \cdot \ldots \cdot [i_{l-1}] \right),$$

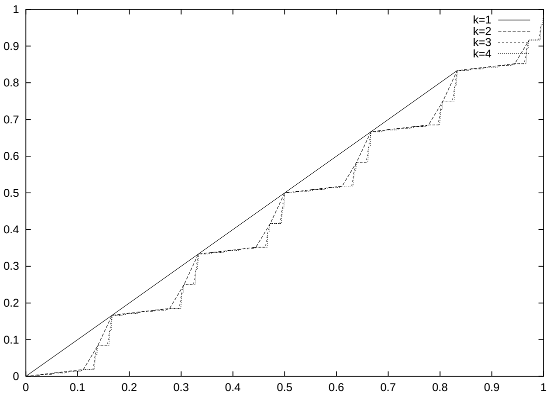
$$\tilde{i_s} = i_s - (\gamma - 2)\langle i_s \rangle.$$

Let $[i_1] = \langle i_1 \rangle = 1$ and for $s \geq 2$ let $[i_s]$ and $\langle i_s \rangle$ be defined as:

$$[i_s] = \begin{cases} 0 & \text{for} \quad i_s = 0, 1, \ldots, \gamma - 3 \\ 1 & \text{for} \quad i_s = \gamma - 2, \gamma - 1 \end{cases}$$

$$\langle i_s \rangle = \begin{cases} 0 & \text{for} \quad i_s = 0, 1, \ldots, \gamma - 2 \\ 1 & \text{for} \quad i_s = \gamma - 1 \end{cases}$$

Figure 3.2 illustrates the values of ψ for $k = 1, 2, 3, 4; n = 2; \gamma = 6$.



Obrázek 5: Values of $\psi(d_k)$ for various k.

3.3. The outer functions ϕ

The functions ϕ_q in equation (47) are constructed iteratively as functions $\phi_q(y_q) = \lim_{r\to\infty}\sum_{j=1}^{r}\phi_q^j(y_q)$. Each function $\phi_q^j(y_q)$ is determined by e_{j-1} at points from the set $\mathcal{Q}^n$. The construction is described in the following.

For $q = 0, 1, \ldots, m$ and $j = 1, \ldots, r$ we compute:

$$\phi_q^j \circ \xi(\mathbf{x}_q) = \frac{1}{m+1}\sum_{\mathbf{d}_k^q} e_{j-1}(\mathbf{d}_k)\,\theta(\mathbf{d}_k^q; \xi(\mathbf{x}_q)), \tag{52}$$

where $\mathbf{d}_k \in \mathcal{Q}$.

The real-valued function $\theta(\mathbf{d}_k; \xi(\mathbf{x}_q))$ defined for a fixed point $\mathbf{d}_k^q \in \mathcal{Q}^n$. The definition is based on a given sigmoidal function σ:

$$\begin{aligned}\theta(\mathbf{d}_k^q; y_q) &= \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q)) + 1) \\ &- \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q) - (\gamma-2)b_k)\end{aligned} \tag{53}$$

where $y_q \in \mathcal{R}$, and b_k is a real number defined as follows:

$$b_k = \sum_{s=k+1}^{\infty}\gamma^{-\rho(s)}\sum_{p=1}^{n}\lambda_p.$$

The functions $\xi(\mathbf{d}_k^q)$ are expressed by equation

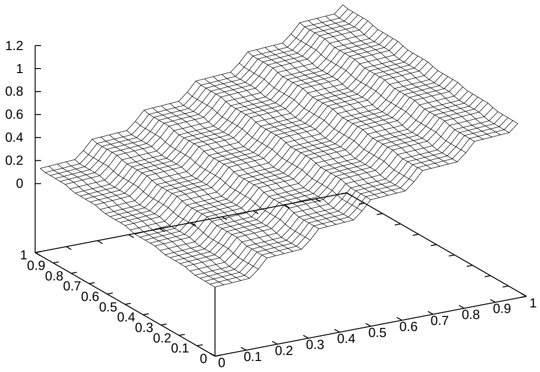
$$\xi(\mathbf{d}_k^q) = \sum_{p=1}^{n}\lambda_p\psi(d_{kp}^q) \tag{54}$$

where $\psi(d_{kp}^q)$ are from (51), and coefficients λ_p are defined as follows.

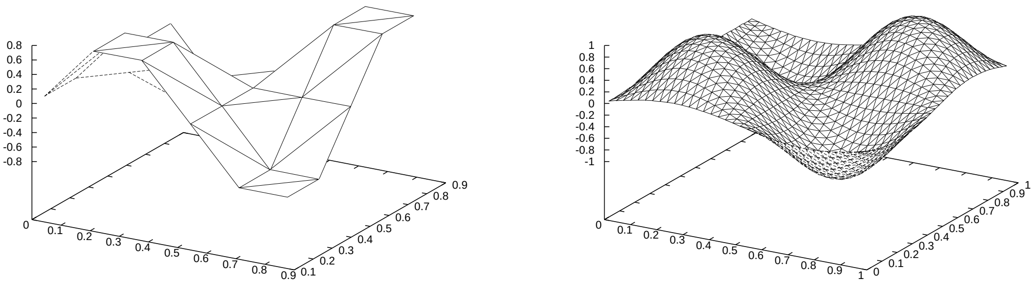
Let $\lambda_1 = 1$ and for $p > 1$ let

$$\lambda_p = \sum_{s=1}^{\infty}\gamma^{-(p-1)\rho(s)} \tag{55}$$

Figure 3.3 shows values of $\xi(\mathbf{d}_k)$ for $k = 2$.



Obrázek 6: Values of $\xi(\mathbf{d}_k)$ for $k = 2$.



Obrázek 7: Approximation of $\sin(6.28x) \cdot \sin(6.28y)$ for $k = 1, 2$.

3.4. Iteration step

Let f is a known continuous function, $e_0 \equiv f$. The r-th approximation error function e_r to f is computed iteratively for $r = 1, 2, \ldots$

$$e_r(\mathbf{x}) = e_{r-1}(\mathbf{x}) - \sum_{q=0}^{m} \phi_q^r \circ \xi(\mathbf{x}_q), \tag{56}$$

where $\mathbf{x} \in \mathcal{I}^n$, $\mathbf{x}_q = (x_1 + q\beta, \ldots, x_n + q\beta)$, and $\beta = \gamma(\gamma - 1)^{-1}$.

The r-th approximation f_r to f is then given by:

$$f_r(\mathbf{x}) = \sum_{j=1}^{r} \sum_{q=0}^{m} \phi_q^j \circ \xi(\mathbf{x}_q). \tag{57}$$

It was shown in [11] that $f_r \to f$ for $r \to \infty$.

4. Ways of parallelization

Neural net like structure of proposed algoritmus provides massive parallelization. We can delegate nodes to compute even such small pieces as d_k. Unfortunately this leads to exponentially increasing number of nodes, and communication between nodes. For such reasons this type of parallelization is not suitable for cluster's computing. We must find new, better level. Feasible candidate can be parallel computation of the members of the sum 57.

In each iteration step r we use $2n+1$ nodes for computing of ϕ_q from inputs r, f_r. The disadvantage of this dividing is the fact, that nodes can't share all d_k. But finally it leads to faster version of learning algorithm we have tried.

In our implementation we use the cluster of workstations Joyce and PVM environment. Approximated functions was two dimensional, and we exhausted five clusters nodes.

References

[1] F. Girosi and T. Poggio. Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation*, 1:461–465, 1989.

[2] Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Procceedings of the International Conference on Neural Networks*, pages 11–14, New York, 1987. IEEE Press.

[3] Věra Kůrková. Kolmogorov's theorem is relevant. *Neural Computation*, 3, 1991.

[4] Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5:501–506, 1992.

[5] A. N. Kolmogorov. On the representation of continuous function of many variables by superpositions of continuous functions of one variable and addition. *Doklady Akademii Nauk USSR*, 114(5):953–956, 1957.

[6] M. Leshno, V. Lin, A. Pinkus, and S. Shocken. Multilayer feedforward networks with a non-polynomial activation function can approximate any function. *Neural Networks*, (6):861–867, 1993.

[7] G.G Lorentz. *Approximation of functions*. Halt, Reinhart and Winston, New York, 1966.

[8] David A. Sprecher. On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355, 1965.

[9] David A. Sprecher. A universal mapping for Kolmogorov's superposition theorem. *Neural Networks*, 6:1089–1094, 1993.

[10] David A. Sprecher. A numerical construction of a universal function for Kolmogorov's superpositions. *Neural Network World*, 7(4):711–718, 1996.

[11] David A. Sprecher. A numerical implementation of Kolmogorov's superpositions ii6. *Neural Networks*, 10(3):447–457, 1997.

Soft Computing Agents and Bang2

doktorand:
Pavel Krušina,
Zuzana Petrová
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O. Box 5, 18207 Prague, Czech Republic
{krusina,petrova}@cs.cas.cz

školitel:
Roman Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O. Box 5, 18207 Prague, Czech Republic
roman@cs.cas.cz

obor studia:
Theoretical computer science

Abstrakt

Enthusiastic for the concept of combining various modern artificial intelligence methods — namely neural networks, genetic algorithms and fuzzy logic controllers — embodied by soft computing area, we are developing a unified software platform, called Bang2. The system serves as a library for many artificial intelligence methods and allows for easy creation of such combinations, and possibly their semiautomated generation or even evolution. It also ensures their deployment through the computer network and parallel processing. Inspired by software agents paradigm we have designed Bang2 as a community of cooperating autonomous software agents.

This paper shows the ideas behind Bang2, together with the brief description of how the system works.

1. Introduction

Since the practical use of artificial life methods like neural networks, genetic algorithms as well as their simple combinations (genetics learning neural net, ...) seem to be widely explored ([1]), we have turned our effort to more complex combinations, which are completely out of focus of scientific community, probably because of the lack of a unified software platform that would allow for experiments with hybrid models.

Design of Bang2 pursues two goals. At first to serve as a library for many artificial intelligence methods and thus help developers to design their own applications. Moreover the unified interface allows to switch easily e.g. between several learning methods and to choose the best combination for application design. Parallel processing is the expected and useful advance here as well as a rapid and easy design. Second goal of Bang2 design involves creation of more complex models, semiautomated models generation and even evolution of models. Evolution of more complex schemata, such as neural network with weights trained by back-propagation and topology trained by genetic algorithm trained by another genetic algorithm and probabilities of genetic operators trained by fuzzy logic controller will constitute the future power of Bang2.

For distributed and relatively complex system like Bang2 it will be favorable to make it modular and to prefer the local decision making against global intelligence, and therefore to take advance of agent technology. Employing software agents also simplifies the implementation of new AI components (unified interface) and addition of them without recompiling and restarting.

Next sections give you a glimpse to the depth of the Bang2. Section 2 describes the overall architecture. Section 3 give you a look-in to the inwards of the environment and the agent. Section 4 will be about agent and agent languages and finally about Bang2 language. Section 5 describes the agent's life cycle and section 6 contains description of the special agents in Bang2. Section 7 outlines our future intentions.

2. Overview of Bang2

Bang2 consists of a population of sundry agents living in the Environment. The Environment provides the necessary support for Bang2's run such as creation of agents, giving them information necessary to survive and be able to communicate (e.g. where are other agents), distribution processes to their computational nodes (parallelism, load balancing). It also delivers messages and transfers data.

Agents are the basic building blocks of Bang2. Each agent provides and requires services (e.g. statistic agent provides statistic preprocessing of data and requires data to process). Agents communicate via special communication language encoded in XML. There are several special agents necessary for Bang2's run (like the Yellow Pages agent maintains information about all living agents and about the services they provide). Other (not special) agents do the real work (read data from files, represent neural net, learn neural net, provide numeric calculation for other agents etc.)

Before any further insight to Bang2 the term agent should be approached. Exact definition of agent doesn't exist, every group using agents provides its own definition. For introduction to software agents see [3]. Generally software agent is a computer program, which is autonomous, reacts to its environment (e.g. to user's commands or messages from other agents) and when nothing interesting happens it doesn't wait for the next event as regular program, but does its own work. It usually follows its own goal. It is adaptive and intelligent in sense that it is able to obtain information it needs by asking somebody (other agent, a human, a server). Moreover it is usually mobile, persistent, and sometimes tries to simulate human character.

Intelligence and mobility of building blocks is one of the essentials and we hope advantages of Bang2. We would like to construct among others RBF net, which looks around if there is any other agent able to learn RBF net's weights or genetic algorithm which sets the suitable genetic operators according the structure of the population to learn.

3. Architecture

As we have said before, we can divide the Bang2 system into two fundamental parts — the environment and the agents. The environment serves as a living space for all the agents, giving them resources they need and serves as a communication layer. These are the main aspects we want to keep on mind when designing and programming the environment:

Abstraction — hiding of raw hardware and OS to our agents and providing most of services and resources in friendly and comfortable manner.

Transparency — hiding as much implementation details as possible and suitable while still allowing agents to explicitly request such informations. The first task the transparency comes to our mind is communication — the goal is to make the communication being simple for the agent programmer and exactly the same for local and remote case while still exploiting all the advantages of the local one.

char*	Sync(Module* agent, char* message)
void	Async(Module* agent, char* message)
Request*	Dsync(Request* req, Module* agent, char* message)
CData*	BinSync(Module* agent, int session, int funcnum, CData* data)
Request*	BinDsync(Request* req, Module* agent, int session, int funcnum, CData* data)
	UFastNX

Tabulka 1: Agent communication functions

Medium	XML strings	CData*	function parameters
Call	Sync	BinSync	UFastNX (hidden by macros)
Generality	High	Agents negotiate at run-time	Hardwired by programmer
Speed	Normal	Fast	The fastest

Tabulka 2: Communication functions properties

Scalability — hope to design and write program with no built-in limits of amount of usable resources. We want our program to be run on computers of very different performance: from small laptops for agents programmers to huge clusters for real number crushing.

Adaptability — ability to run agent schemas developed on small systems on huge ones and vice versa. Preferably with only small need of manual interference.

Helper functions — being friendly to agent programmer. Insert a lot of functionality to agent base class and provide a code generators.

3.1. Communication layer

What we call communication layer is mainly the environment and a small code in agent base class. Purpose of it is to allow communication between agents. What we expect from it:

Simplicity — we want the communication to be simple from the agent programmer's point of view, something like a single function call.

Location transparency — there should be no difference for the agent programmer between communication to local and remote agent.

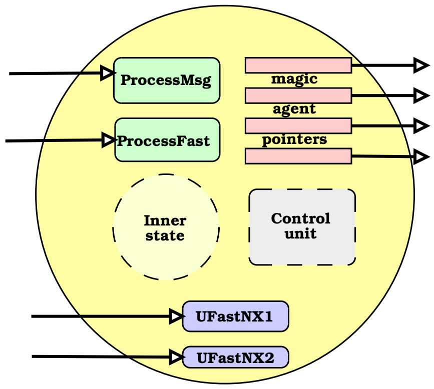
Synchronicity – we want to provide an easy way how to select synchronous, asynchronous or deferred synchronous mode of operation for any single communication act.

Efficiency — we want to be efficient both in passing XML strings and binary data.

As the best abstraction for the agent programmer we have chosen the model of object method invocation. From its advantages let us mention the facts that programmers are more familiar with concept of function calling then message sending and that the model of object method invocation simplifies the trivial but much common cases while keeping the way to the model of message passing open and easy. The communication functions visible to agent programmer can be found in table 1 and their characteristics in table 2. Sync is a blocking call of the given agent returning its answer, Async is non-blocking call discarding answer and Dsync is non-blocking call storing answer at negotiated place. BinSync and BinDsync are same as Sync and Dsync but the exchange binary data instead of XML strings. UFastNX is a common name for set of functions with number of different parameters of basic types usually used for proprietary interfaces.

3.2. Agents

All agents in Bang2 are regular C++ classes derived from base class Agent which provide common services and connection to environment (Fig. 8). Each agent behavior is mainly determined by its ProcessMsg functions which serves as main message handler. The ProcessMsg function parses



Obrázek 8: Agent inwards

```
TRIGGER1( request_setfriend, id )
{
  SETAGENT( friend, id );
  OK;
}

TRIGGER0( request_pingfriend )
{
  return Sync( friend,
     "<request><ping/></request>" );
}
```

Obrázek 9: Triggers code

the given message, runs user defined triggers via RunTriggers function and, if none is found, the DefaultBehavior function is called. The DefaultBehavior function provides standard processing of common messages. Agent programmer can either override ProcessMsg function on his own or (preferably) write trigger functions for messages he want to process (Fig. 9). Triggers are functions with specified XML tags and attributes. RunTriggers function calls a matching trigger function for a received XML message and fills up the variables corresponding to specified XML attributes with the values (see 4).

Magic agent pointer is in fact an association of a regular pointer to Agent object with a string containing its stringified handle registered to the Agent class, so DefaultBehavior function can automatically adjust the pointer and the handle according to information emitted by Black Pages.

Finally inner state is a general name for values of relevant member variables determining the mode of agent operation and its learned knowledge. The control unit is its counterpart — program code manipulation with the inner state and performing agent behavior, it can be placed in all ProcessMsg/ProcessFast functions or triggers.

4. Communication language

Consider a simple example that iterates various stages in a design of an agent communication. Let's have a neural net. It looks around for a learning agent, negotiates with other agents if they are able to learn it, are free (do not learn any other agent already) and negotiates format of data to transfer. Then these two agents connect together and exchange data (neural nets weights and error). We need to:

- **specify message headers**. Should be human readable.

Special messages — only Yellow Pages understands them:

```
<broadcast><halt/></broadcast>
<inform><created myid="!000000000001" name="Lucy"
                 type="Neural Net.MLP"/></inform>
```

Reply from Yellow Pages:

```
<ok>Agent Lucy, id=!000000000001, type=Neural Net.MLP created</ok>
```

All agents understand this message:

```
<request><ping/></request>
```

Obrázek 10: Example of Bang2 language for agent negotiation

- **language for agent messages** (negotiation, control sequences). Should be human readable, declarative.
- **language for data transfer**. Should be able to transfer complex data structures through simple byte stream.

There are several languages for these purposes. ACL ([4]) and KQML ([2] — widely used, de facto standard) define (among others) format of message headers and communication protocols. They are lisp-based.

KIF (KQML group — [5]), ACL-Lisp (ACL group — [4]) are languages for data transfer. They both came out of predicate logic and both are lisp-based, enriched with keywords for predicates, cycles etc. XSIL [8] and PMML [7] are XML-based languages designed for transfer of complex data structures through the simple byte stream.

Messages in Bang2 system are syntactically XML strings. Headers are not necessary, because the inner representation of messages (method invocation), so the sender and receiver are known. First XML tag defines the type of the message (similar to message type defined in an ACL header). Available message types are: *request, inform, query, ok* (reply, no error), *ugh* (reply, an error occurs).

The rest of the message (everything between outermost tags) is the content. It contains commands (type request), information provisions, etc. Some of them are understandable to all agents, others are specific to one agent or a group of agents.

There are two ways how to transfer data:

- **As a XML string** — human readable, but lack performance (the lack of performance is not fatal in agents' negotiation stage (as above), but is a great disadvantage when they're transferring data).
- **As a binary** — much quicker, but receiver have to be able to decode it.

Generally in Bang2 the XML way of data transfer is implicit and the binary way is possible after the agents make an agreement about the format of transferred data.

5. Life cycle

Now we look at the agent life cycle and focus on what the agent really is, how is it brought to life, what actions can it perform during its life and finally how it dies.

```
<request><getvector row="45"/></request>
<request><getvector/></request>
<ok><data separator=",">Here are binary data</data></ok>

<query><bin><request><getvector/></request></bin></query>
<ok session="5" funcnum="1"/>
```

Then the communication is completely binary, realized with function calls.

```
BinSync(...)
```

Obrázek 11: Example of Bang2 language for data transfer

Agent code is stored in form of shared library binaries prepared for loading on demand. Agent memories and initial states are stored and retrieved via White Pages which serve as a database for agent inner states and object repository. White Pages also provide a search across all possible agent service. Wanting to create an agent, one can send a message to a Launcher agent to create a new agent of specified type. The first thing a newly created agent does is receiving the request init message. The agent can do all initializations here and should also send an inform created message to Yellow Pages — the database of all living agents. Yellow Pages in turn gives a human friendly name to agent and store information about services it provides. When an agent needs to find some other, it can use Yellow Pages and query an information.

When an agent learned anything worth, it can store it into the White Pages in order to make it accessible by all future incarnations of the same agent type. Of course agents can retrieve information they stored to White Pages. And finally as all of us, even our agents die. They cooperate with Black Pages to let the others know about it to avoid call of non-existing agents code.

6. Agents

In this section we will describe the most important special agents: Black Pages, Launcher, Yellow Pages, White Pages and then give you an example of an AI agent — an agent representing generic genetic algorithm.

6.1. Black Pages

Black Pages is an agent responsible for correct processing of agent death. The need of Black Pages have arisen from our thoughts upon agents destruction. If an agent dies while others keep a pointer to it and may call it, we must to find a way of signaling the agents death to other to avoid of calling destroyed agent code. We found two different ways: First, at the moment of agent death do a broadcast telling this fact to all. Second: leave some residual object (we call it zombie) in the place of dead agent and let it answer an error message for all requests form others. The first is slow, the second memory inefficient. We decided to combine the two in the following manner: If an agent dies, it leaves zombie and signals its death to Black Pages. Black Pages in turn in some time intervals do a broadcast with all dead agents list and finally destroy the zombies. By tuning the time interval and dead list maximal length we can easily change the speed/used memory ratio. In fact Black Pages cannot destroy any agent or zombie directly since the only agent with legal connection to these environment function is Launcher, so Black Pages calls it for the real dirty work. Similarly Black Pages has not its own timer functions but uses the Cron agent for them and uses Yellow Pages for broadcasting. The way agent dying is implemented has much in common with agent moving, upgrading and cloning, so Black Pages is used also in these cases. Also for the messages broadcasted by the Black Pages is provided a default behavior which automatically adjusts the magic agent pointers. Magic agent pointer is something as a regular agent pointer but registered to be automatically updated.

6.2. Launcher

Launcher is an agent providing special environment (more exactly said airport) functionality in agent-like manner. It name comes from it main service — launching new agents into life. It also can destroy them or halt the whole airport. For the tight connection between Launcher agent and the environment's airport, there must be an one to one correspondence between them.

6.3. Yellow Pages

Yellow Pages maintains information about all agent currently existing in the Environment — that means agents that are already invoked by a Launcher. In the opposite White Pages contains information about all agent type, which can be invoked. Yellow Pages knows about each agent its

- **name** — human comprehensible name, e.g. lucy, david, alice, etc. Is unique among all agent in Yellow Pages' database,
- **id** — 12 hexadecimal digits, encoding some internal information, e.g. on which computer the agent resides, etc., unique inside the Yellow Page s's database,
- **type** — specify a function of special agent or a (AI or other) functionality provided by an agent — e.g. launcher, console, MLP (multi layer perceptron), FLC (fuzzy logic controller), etc.
- **connections** to other agents. To which agents is this agent connected. And character of the connection (input/output, learning, error function, some auxiliary agent — provides complex numeric calculations, conversion between something, etc.)

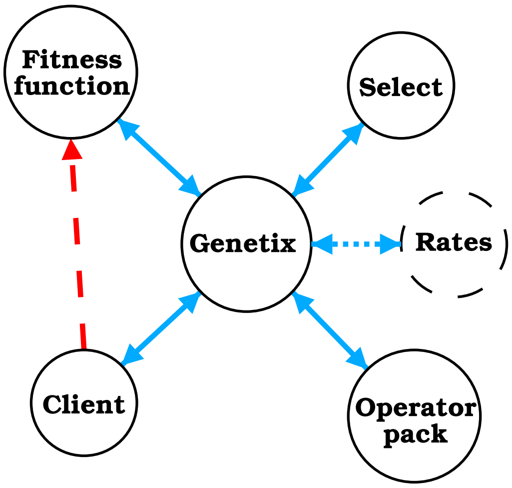
Each agent immediately after creation sends to Yellow Pages information about its name, id and type. Yellow Pages is then able to answer arbitrary queries about name, id and type: e.g. translates between names and ids — serves as a name server, or returns id (or names) of all agents of specified type (and thus able to provide specified function), broadcasts to all agents or to agents of specified type. It is also gathering information about connections between agents and is able to answer queries about connections.

6.4. White Pages

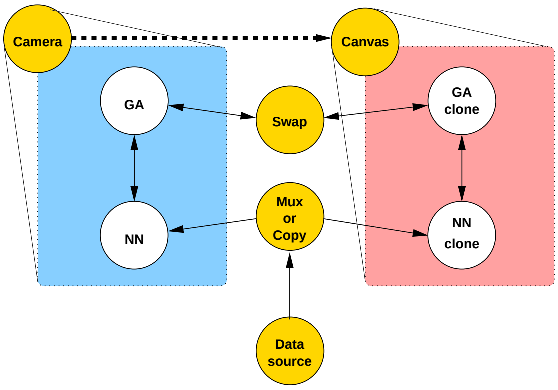
White Pages maintains information about all agent types. It is partially unimplemented so carefully: it contains information about the name of the agent type (same as the type of agent in the Yellow Pages' database). Then for each agent it contains information about structure of the type (it will be used when transferring data) and moreover contains information about configurations of agents (agent persistence).

6.5. Genetix

The goal is to create as generic and universal genetic algorithm engine as possible and even little more. For this reason the GA of our has neither the fitness function or genetic operators hardwired into it but rather uses external agents as a plug-ins for these functionalities. Now Genetix (how our genetic algorithm agent is called) exploits the outer agents for fitness function, genetic operators, selection method and genetic operators rates tuning (Fig. 12). The Genetix is able to let the genetic operators, selection method and genetic operators rates tuning be set in advance from the client or anyone else, let them be set on the actual teach request or if any or all of them are not set when really needed look for the suitable agents on its own via Yellow Pages or White Pages. So when for example neural network wants to be taught it can simply ask Genetix to teach its weights and the Genetix will in turn look up the genetic operator package for floating point numbers vectors and a basic selection method — roulette wheel. When we want to use Genetix for anything else, we only need a fitness function evaluating agent — that is definition of a problem — and a genetic operators package — that is a way of manipulation with genoms or in other words a definition of how to get from a set of problem solution candidates to candidates modified in some way.



Obrázek 12: Genetic algorithm implementation scheme



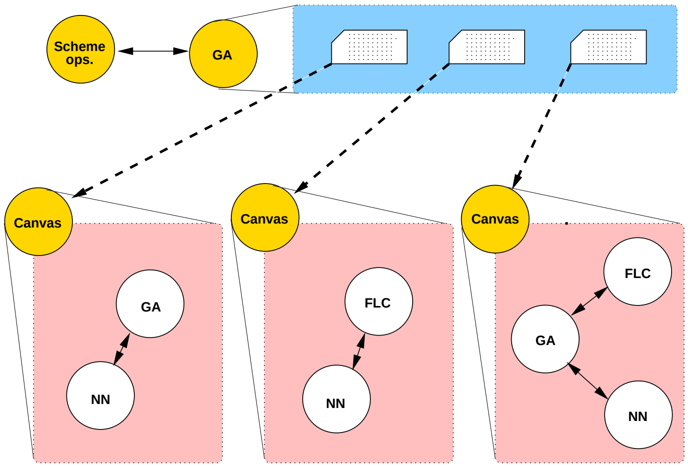
Obrázek 13: Task parallelization

7. Conclusion

For now, the design and implementation of the environment is complete. Agent class as a base class for all agents is ready to use and provides a set of useful services while still getting new features. Inter-agent communication language is near to become stable. So for now, we are to create a set of agents of different purpose and behavior to be able to start designing and experimenting with adding real agentness to the system. We have a simple script called GenAgent to help agent programmers by generating a empty skeleton of new agent class. Actually the work on the agents has started, there are going to be GA and RBF agents in near future.

For experimenting on agent schemes, we need agents of various types. We want to try mirrors, parallel execution, automatic scheme generating and evolving. Also concept of an gent as the other agent's brain by means of decisions delegating seems to be promising. Another thing is the design of load balancing agent able to adapt to changing load of host computers and to changing communication/computing ratio. To make interaction with human more comfortable we want to create a user-friendly graphical user interface. Preferably in way allowing easy swap to non-graphical representation. And finally we think about some form of inter Bang2-sites communication.

In the following we discuss some of these directions in more details.



Obrázek 14: Scheme evolving

7.1. Task parallelization

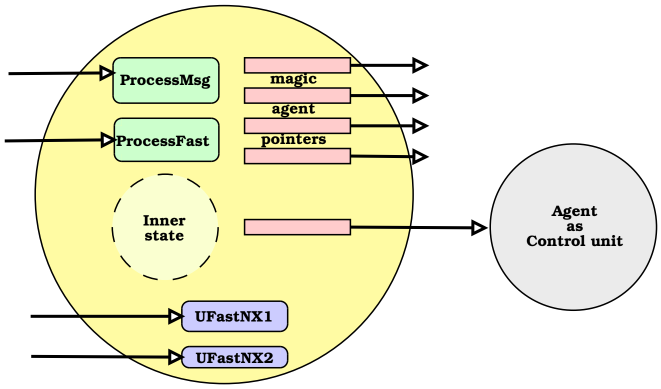
There are two ways of parallelization: by adding ability to parallelize its work to a computation agent or by creating generic parallelization agent able to manage non-parallel agent schemes. Both have their good and weak sides, but here is no reason not to implement both and let the user or agent programmer to choose. Consider an example of a genetic algorithm. It can explicitly parallelize by cloning fitness function agent and letting the population being fitnessed simultaneously. Or on the other hand, the genetic algorithm can use only one fitness function agent, but be cloned together with it and share the best genoms with its siblings via a special purpose genetic operator. We can see this in figure 13, where agents of Camera and Canvas are used to automatize the subscheme-cloning. Camera looks over the scheme we want to replicate and produces its description. Canvas receives such description and creates the scheme from new agents. You can imagine cases where each of the above approaches is better then the other, so it make sense to defer this decision till the real task is considered.

7.2. Agents scheme evolving

When thinking about implementing the task parallelization, we found very useful to have a way of encoding scheme descriptions in way understandable by regular agents. Namely we think about some kind o XML description. This leads to idea of agents not only creating and reading it, but also manipulating with it. All we need to be able to evolve agent schemes by generic genetic algorithm is to create a suitable genetic operator package. You may ask, what will be the fitness function for such genoms. The answer is simple: the part of generic task parallelization infrastructure (namely the Canvas, see fig. 14). For genetic evolving of schemes we can use the Canvas for testing newly modified schemes. In fact the only thing we want to add to be able from task parallelization advance to scheme evolving is the actual scheme genetic operator package. I find this a nice proof of reusability and good design of Bang2.

7.3. Agent as a brain of other agent

As it is now, the agent has some autonomous - or intelligent - behaviour encoded in standard responses for certain situations and messages. A higher degree of intelligence can be achieved by hard-coding some consciouseness mechanisms into agent. One can think of creating a planning agents, Brooks subsumption architecture agents, layered agents, or Franklin "conscious" agents. We plan to create a universal mechanism via which a standard agent can delegate some or all of its control to a specialized agent that serves as its external brain. This brain can independently seek for supplementary information, create its own internal models, etc, and finally advise the original agent what to do.



Obrázek 15: Agents as brains

References

[1] P. Bonnisone, "Soft computing: the convergence of emerging reasoning technologies", *Soft Computing*, vol.I, pp. 6–18, 1997.

[2] Tim Finnin, Yannis Labrou, James Mayfield, "KQML as an agent communication language", *Software Agents*, MIT Press, Cambridge, 1997, http://www.cs.umbc.edu/agents/introduction/kqmlacl.ps.

[3] Stan Franklin, Art Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *Intelligent Agents III*, pp. 21–35, 1997.

[4] Foundation for Intelligent Physical Agents, "Agent Communication Language", *FIPA 97 Specification*, 1997, http://www.fipa.org.

[5] Michael Genesereth, Richard Fikes et. al., "Knowledge interchange format, version 3.0 reference manual", Technical Report, Computer Science Department, Stanford University, 1992. http://www.cs.umbc.edu/kse/kif/.

[6] Roman Neruda, Pavel Krušina, "Creating Hybrid AI Models with Bang", *Signal Processing, Communications and Computer Science*, pp. 228–233, 2000.

[7] "PMML v1.1 Predictive Model Markup Language Specification", Technical Report, Data Mining Group, 2000, http://www.dmg.org/html/pmml_v1_1.html.

[8] Roy Williams, "XSIL: JAVA/XML for Scientific Data", Technical Report, California Institute of Technology, 2000.

Cesty a úskalí imunitní sítě buněk

doktorand:
PŘEMYSL ŽÁK
ÚI AV ČR Pod Vodárenskou věží 2, 182 07 Praha 8
zak@cs.cas.cz

školitel:
MARCEL JIŘINA
ÚI AV ČR
marcel@cs.cas.cz

obor studia:
Výpočetní technika - Neuronové sítě

Abstrakt

Přístupů k problematice využití imunitních principů k výpočtům je v současnosti již celá řada. Ve svém studiu jsme se zaměřili na přístup formálně vycházející z oboru umělých neuronových sítí. Výsledkem je však síť připomínající IBMs (Individual Based Models) známé z biologie. Popis takových sítí je často do značné míry empirický a může třeba někomu připadat i jako neplodné hraní. Koneckonců, posuďte sami.

1. Zařazení

Na počátku byla myšlenka zobecnit do značné míry strukturně neměnné neuronové sítě na síť buněk pohybujících se ve vymezeném prostoru a komunikujících lokálně. Po značném programátorském úsilí se zdařilo postavit funkční model. Popis modelu však naráží na mnohá úskalí, navíc kontrola popisu se skutečným výstupem je komplikována přítomností náhodného prvku - pohybem buněk.

Z literatury se zdá být zřejmé, tento IBM (Individual Based Model) přístup z výše uvedených důvodů není příliš populární. Snad jen pro ilustraci uvedu práce ukazující směry dosavadního vývoje umělých imunitních sítí. Představme si tedy prostor, kde se pohybují buňky schopné mezi sebou komunikovat, dělit se a zanikat. Každá z nich může být jedinečná a rozlišitelná.

- Alan Perelson [1] koncem 80. let představil myšlenku "tvarového prostoru" (shape space), která mu posloužila jako argument podporující postulát úplnosti repertoáru imunitního systému (tj. že imunitní systém je v pripncipu schopen rozpoznat jakýkoli vzor).

  Perelson uvažuje, že tvar je popsán n-ticí reálných čísel z omezeného intervalu (rýhy a boule povrchu). Jestliže vzor který má být rozpoznán je rovněž z omezeného intervalu n-rozměrného intervalu a buňka je schopná rozpoznat vzor s nějakou odchylkou e , potom nutně stačí konečný počet buněk k pokrytí uvedeného intervalu vzorů. (za předpokladu rovnoměrného rozložení bodů tvarového prostoru)

  Co se týče paměti, Perelson uvažuje dva možné přístupy - statický a dynamický. U statického přístupu se daný vzor zapamatuje prostřednictvím buněk daný vzor rozpoznávající. Tyto buňky mají pak dlouhou životnost. U přístupu dynamického, je informace o vzoru uložena do cyklu reakcí buněk, které se tímto stále udržují v pohotovosti. Dalo by se říci že první přístup je možno chápat jako část druhého s tím, že paměťové buňky nevyžadují neustálou aktivaci okolí, jako je to u všech ostatních buněk.

- Perelson nastínil spíše obecné otázky rozpoznávání vzoru a funkce sítě buněk. De Boer [3] pak dosti podrobně popisuje konkrétní model založený na dalších biologických inspiracích. Přidáním dalších podmínek se vše dosti komplikuje a proto jeho model, obsahuje pouze dva klony buněk. (Buňky jednoho klonu jsou schopné rozpoznat jen buňky z druhého a naopak.)

  Model je tvořen soustavou dvou diferenciálních rovnic popisujících přísun nových buněk, jejich zanikání, a dělení aktivovaných buněk (tzv. B model). K těmto dvěma rovnicím jsou pak přidány další rovnice, popisující dynamiku protilátek, které jsou těmito klony produkovány (tzv. AB model).

  Popsat chování podobných modelů je snadné pouze v jednoduchých případech. Složitější modely však jsou ve své podstatě nestabilní, v nejlepším případě oscilují kolem nějaké hodnoty.neobsahují jeden stabilní stav. Zároveň však podstatně více odpovídají skutečnosti, kdy počty buněk jednotlivých klonů oscilují - často chaoticky. Zhang [4] například zkoumá závislost chaotického chování (Ljapunovův exponent) De Boerova AB modelu. Ukazuje se že i malá změna některých parametrů může způsobit změnu charakteru oscilací.

- Třetím pohledem na imunitní systém je pohled zvenčí. V této oblasti vzniká nejvíce aplikací například v oboru řízení nebo rozeznávání defektů, případně virů. Zde se bere v úvahu celková strategie imunitního systému včetně součinnosti jednotlivých orgánů.

  Součástí mechanismů těchto komplexních umělých imunitních sítí jsou často i genetické algoritmy. Výsledky jsou pak porovnávány s neuronovými sítěmi, fuzzy regulátory či "čistými" genetickými algoritmy.

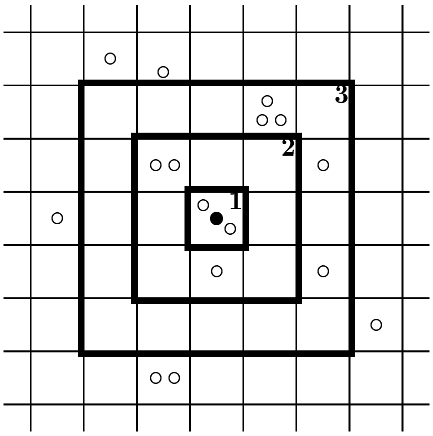
- Teď by měl následovat odstavec o IBM přístupu. Bohužel se nám nepodařilo v oblasti umělých imunitních sítí cokoli podobného najít. Snad aspoň článek srovnávající chování klasického IBM u nás známého jako "lišky-zajíci" ve dvou režimech komunikace: lokálním a globálním [8]. V prvním případě komunikují subjekty pouze ve svém okolí a ve druhém pak všichni se všemi. Stabilní stavy i chování za dostatečně dlouhou dobu byly shledány v obou režimech shodnými, což umožňuje poněkud zjednodušit popis takovéhoto systému. I tento výsledek je však z větší části založen na empirii.

2. Trocha popisu

Nyní bude asi třeba osvěžit paměť čtenáře a připomenout alespoň základní principy modelu [6, 7].

Stavba sítě:
Celá síť je tvořena plochou (třeba čtvercem) skládající se z malých plošek (čtverečků), které představují místa výskytu buněk. Počet buněk přítomných najednou každém takovém místě je omezen.

Buňky se pohybují v jednotlivých krocích po síti. K každém kroku je prozkoumáno okolí buňky, z kterého může buňka přijmout signál.



Obrázek 16: Zóny působení buňky

V daném kroku buňka nejprve zpracuje signál pomocí vztahu

$$INP_i = \sum_{j=1}^{S} w_{i,j} X_{i,j} - THR_i \tag{58}$$

INP_i	*vstup i-té buňky*
S	*počet buněk v okolí*
$w_{i,j}$	*váha vztahu mezi i-tou a j-tou buňkou*
$X_{i,j}$	*signál přicházející od j-té do i-té buňky*
THR_i	*práh i-té buňky*

Dále pak použije svou aktivační funkci. V praxi se nejlépe osvědčila zvonovitá funkce vzniklá odečtením dvou sigmoid, která svým tvarem nejlépe vystihuje biologickou skutečnost [2, 5].

$$DSigm(INP, posunuti) = Sigm(INP) - Sigm(INP - posunuti) \tag{59}$$

Některé parametry sítě:

PRO CELOU SÍŤ
I x J - rozměry sítě
K - maximální počet buněk na jednom místě
ampl - zesilovací konstanta signálu v síti
omezený prostor působí změnu hustoty (intenzity) signálu při měnícím se počtu buněk

PRO KAŽDÝ TYP ZVLÁŠŤ
max, steep - parametry aktivační funkce
W_i - váhový vektor
klasické parametry neuronu, matice vah je vždy násobena ještě konstantou ampl

PRO JEDNOTLIVÉ BUŇKY
i, j - pozice buňky
parametr ovlivněný náhodou a působící šum

Činnost sítě:
Nejprve se síť inicializuje. Nastavují se počáteční hodnoty parametrů platných pro celý model, pro jeden typ buněk i pro jednotlivé buňky.

Dalším krokem je příjem signálu z okolí buňky a jeho zpracování popsané výše.

Výsledek tohoto zpracování se pak porovnává s mezemi pro dělení a zánik. Je li výsledná aktivita příliš malá, buňka zaniká, je-li naopak dostatečně vysoká, buňka se rozdělí.

Nakonec se buňky přemístí na své nové místo. Přesun je náhodný maximální vzdálenost přesunu je jedním z parametrů.

3. Výsledky

Vývoj této sítě probíhal klikatými cestičkami. V První fázi se podařilo formalizovat imunitní principy do sítě buněk připomínající neuronovou síť. Fáze ladění byla spojena s výběrem a studiem parametrů sítě. Na tomto základě se podařilo síť stabilizovat do stabilního oscilačního stavu. Tento stav byl robustní vůči zásahům zvenčí, avšak dosti citlivý na nastavení parametrů.

Následoval pokus definovat a vyzkoušet učení. Učení bylo definováno jako schopnost sítě zareagovat na známý signál efektivněji a rychleji se uvést do rovnovážného oscilačního stavu. Proběhly pokusy s několika učícími parametry.

Vývoj se prozatím odehrával pouze na empirické bázi, proto jsme se pokusili alespoň částečně popsat síť matematicky. Vliv matice vah na stabilitu sítě se však zatím nepodařilo uspokojivě vysvětlit.

Dále jsme se snažili popsat síť rekurentním vztahem vycházejícím ze zjednodušených principů algoritmu sítě. V zájmu usnadnění popisu bylo třeba vše maximálně zjednodušit. Zanedbali jsme i to ze komunikace je lokální a vyjádřili aktivitu buňky (příp. klonu buněk) vztahem

$$ACT_i^{(t)} = f(\sum_{j=1}^{S} w_{i,j} ACT_{i,j}^{(t-1)} - THR_i) \tag{60}$$

Zdánlivě totožné se stavem neuronu v neuronové síti. Stav buňky však není jediným nositelem signálu. Lépe řečeno, je-li aktivita dostatečně vysoká, buňka se rozdělí a celý klon se posílí. Opačně je to pak, je-li její aktivita pod mezí zániku. My pak můžeme sledovat změnu počtu buněk v jednotlivých klonech, které jsou nutně závislé na předchozí aktivitě.

$$N_i^{(t)} = f(\sum_{j=1}^{S} w_{i,j} ACT_j^{(t-1)} - THR_i) \tag{61}$$

Zde však již není tak docela jasné jaká je funkce f. Navíc zpracujeme-li reálné výstupy sítě, zjistíme, že vztah by měl být posunut o krok dozadu (t-2):

$$\mathbf{N_i^{(t)}} = \mathbf{f}(\sum_{\mathbf{j=1}}^{\mathbf{S}} \mathbf{w_{i,j} ACT_j^{(t-2)}} - \mathbf{THR_i}) \tag{62}$$

Navíc aktivita buňky musí souviset s počtem buňěk v minulosti.

Je vidět, že celý systém má určitou vazbu na minulost. Proto má smysl hledat vztah ve tvaru

$$N_i^{(t)} = N_i^{(t-1)} + N_i^{(t-1)} * Konst * f(\sum_{j=1}^{S} w_{i,j} ACT_j^{(t-z)} - THR_i) \tag{63}$$

kde N_j je počet buněk v jednotlivých klonech a z je zpoždění reakce.

Zdá se však, že systémy s tímto popisem divergují (příp. konvergují k nule, tj. k vymizení sítě) pokud uvažujeme běžné funkce f. Například při použití klasické sigmoidy systém, který ve skutečnosti osciluje kolem nějaké hodnoty, by měl teoreticky exponenciálně růst.

Podstatnou roli zde hraje pravděpodobně již zmíněná proměnlivá hustota signálu v souvislosti s disproporcí zpožděné aktivity a momentálním počtem buněk.

4. Závěrem

Takto zjednodušená buněčná síť ukazuje, že zvolíme-li vhodně váhy síť je stabilní a nezávisí na počátečních hodnotách. Pomocí empirického popisu chování sítě můžeme navrhnout rekurentní vztah popisující dynamiku sítě. Zajímavou vlastností sítě je vnitřní paměť jednotlivých klonů buněk, která způsobuje obtíže v matematickém popisu, avšak zjevně přispívá spolu s omezeným prostorem ke stabilitě sítě.

References

[1] Perelson, A.S.: Immune network theory. *Immunol.Rev.* **110**, 5-36 (1989)

[2] Stites, D.P., Terr, A.I.: *Basic and Clinical Immunology.* Applenton & Lange, A Publishing Division of Prentice Hall, (1991)

[3] DeBoer, R.J., Perelson, A.S., Kevrekidis, I.G.: Immune network behavior I. and II. *Bull. math. Biol.* **54** 745-816 (1993)

[4] Zhang, L., Du, Ch., Qi, A.: Complex behaviours of AB model describing idiotypic network. *Bull. math. Biol.* **56** 323-336 (1994)

[5] *Pokroky v imunologii.* Univerzita Karlova, Sekce postgraduálního studia v biomedicíně (1994)

[6] Žák, P.: Immune Network - Model Inspired by Immune System. *Neural Network World*, Vol.7, **6**, 739-756 (1997)

[7] Žák, P.: Cell Network - Dynamic Neural Network. Learning Problems. *In: EUFIT'98. 6th Europien Congress on Intelligent Techniques and Softcomputing*, Vol.: 1. - Aachen, Mainz 1998, 323-325 (Held: EUFIT'98, Aachen, DE, 98.09.07-98.09.10)

[8] Caswell, H., Etter, R.: Cellular autometon models for competition in patchy environments: Facilitation, inhibition, and tolerance. *Bull. math. Biol.* **61** 625-649 (1999)

Information theoretical approach to support medical decision making using electronic patient records

doktorand:
Ing. Petr Hanzlíček
EuroMISE Centrum UK a AV ČR, Pod vodárenskou věží 2, 180 00, Praha 8
hanzlicek@euromise.cz

školitel:
Prof. RNDr. Jana Zvárová, DrSc.
EuroMISE Centrum UK a AV ČR, Pod vodárenskou věží 2, 180 00, Praha 8
zvarova@euromise.cz

obor studia:
Teoretická informatika

Abstrakt

In this paper we describe new tools developed in the frame of the project I4C-TripleC of the 4th Framework Programme for easy and flexible entering of patient data. This program named ORCA (Open Record for Care) uses structured data entry and storage, which enables easy processing of entered data. Several information measures and algorithms for extracting the optimal variable set for decision making at the minimal costs are also described. This approach is based on information measures of stochastic dependence and conditional stochastic dependence and methods of information theory.

Data acquisition using electronic patient records

Patient-record data (history, physical examination, laboratory data, prescribed drugs, diagnoses, etc.) and cardiac signals (ECGs, blood pressure curves) nowadays are mainly collected in paper records and folders. Such folders are available at one place only and frequently are not available when required by the clinicians. New approaches for data collecting, storing and using for further research were developed in the framework of the European project I4C. The data are transmitted to care providers in an electronic and integrated manner by using multimedia workstations. All data may then be accessed through these workstations located in practices, consultation rooms, outpatient clinics, preferably interconnected and integrated through computer networks. Then such data in principle can be used for direct patient care, but also for quality assessment of care, research and education or management and planning.

The project I4C of the 4th Framework Programme (1996 - 1998) was carried out for the further advancement of cardiac care. It was focused on clinical applications and its main goals were as follows.

- integrated access to data wherever stored;
- support of evidence-based care by remote electronic consultation and peer review;

- more comprehensive and more consistent recording of patient data, images, videos and bio-signals, all combined in a multiple patient record.

With the support of the I4C project the new approach for multimedia electronic patient record has been developed. Multimedia patient record ORCA (Open Record for CAre) is a system that integrates a possibility of structured patient data entry including history, medication, symptoms and more with multimedia objects as ECG, angiography or laboratory data [11]. Data can be entered either using prepared custom forms for special purposes or directly using a knowledge tree. The main advantage of the data model implemented in Orca is the structured data storage which makes possible to translate the entered data easily to other languages, to further process the data using statistical methods or to show entered data according to specific user needs. When it is impossible to enter determined facts using information in a knowledge tree, Orca provides a possibility to insert free text entry into the patient record. However, the prefered way is the structured data entry. Current version of Orca includes a knowledge tree and user interface translation into eight languages. The Orca system uses client-server model, which gives good performance, security and scaleability. For small installations like general practicioner's office, Orca can be used on a single computer with Windows95 using locally installed SQL server. The project I4C-TripleC intends to validate integrated workstations and ORCA system in three hospitals in Central Europe (Prague, Bratislava and Caslav) to support the continuity of cardiac care.

Decision making support using information theory tools

Special problem in decision making occurs when a decision maker has too much empirical information at his/her disposal. Mostly it is large database of observations where many variables are recorded. The aim of decision maker is to reveal variables that will bring him/her quickly sufficient information for decision making at the minimal costs. In fact, it is a special case of a general problem of choice of a relevant piece of information for decision making.

Let us specify our assumptions in more detail. We denote $X = \{X1, X2, ..., Xr\}, r \geq 1$, a set of variables (*independent variables*) than can be *quantitative* (e.g. weight, temperature) or *qualitative* (presence or absence of certain factor). The vector of independent variables X has finitely many possible values $x \in X$ occurring with probabilities $p_X(x) > 0$. Each subset of independent variables $X' \subset X$ is assigned with certain nonnegative cost $c(X') \geq 0$, describing the cost of obtaining the values of the combination of variables X'. It can reflect monetary expenses, nevertheless, mostly it expresses the cost in more general way. For example, in a special case of medical decision making, invasive methods of obtaining value of symptom variables can be painful or risky, therefore the cost c(X') should be high. We denote $Y = (Y1, Y2, ..., Ys)$ a set of variables describing possible decisions (dependent variables), $s \geq 1$. Our task is to find a set of symptom variables X', called set of independent variables, such that variables X' make it possible to estimate with high credibility possible decisions Y. In the process of searching of variables in X' we should minimize the total cost of selected set of variables X'.

Information measures of stochastic dependence

Our approach for extracting relevant information from database is based on information measures of stochastic dependence and conditional stochastic dependence and methods of information theory, described in [1], [2], [3], [4], [5]. Roughly speaking, information measures are nonnegative numerical characteristics of strength of stochastic dependence between two variables (respectively the strength of conditional dependence between two variables given values of the third variable). They have been developed and studied in information theory as tools to estimate Bayes risk. Important properties of measures of stochastic dependence have been pointed out by A. Perez [6], [7] one of the founders of the Czech school of information theory. Mainly the measures of dependence based on Shannon's information were studied, but also measures based on general concept of f-information were proposed by I. Vajda [5]. The concept of multiinformation, introduced as a measure of simultaneous dependence, was studied by M. Studeně [8], [9]. It was shown that multiinformation has close connection to conditional Shannon's mutual information, which serves as a measure of conditional stochastic dependence.

Let us denote $H(X)$ *entropy* of independent variable X, calculated as

$$H(X) = -\sum_{x \in X} p_X(x) \ln p_X(x)$$

and $H(Y)$ *entropy* of dependent (decision) variable Y, calculated as

$$H(Y) = -\sum_{y \in Y} p_Y(y) \ln p_Y(y)$$

and I(X;Y) *Shannon mutual information* calculated as

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p_{XY}(x,y) \ln \frac{p_{XY}(x,y)}{p_X(x) p_Y(y)}$$

where $p_{XY}(x,y)$ is the joint probability distribution of X,Y, $p_X(x)$ and $p_Y(y)$ are corresponding marginal distributions.

According to well known Shannon inequality $0 \leq I(X;Y) \leq min\{H(X), H(Y)\}$ and therefore *Shannon's information measure* of stochastic dependence Y on X is defined as

$$\delta(Y \mid X) = \frac{I(X;Y)}{H(Y)}$$

Further we will call this measure as the *influence of X on Y.*

Our task is to find if it is possible to make justified decision described by decision variables Y using independent variables X. If yes, we should find a relatively small set X' of variables from the set of independent variables X where strong stochastic dependence between variables X' and variables Y is seen. It allow us to estimate with high credibility the values of decision variables in Y (set of dependent variables) on the basis of independent variables in X'. The choice of X' should take into account the cost of obtaining variables, i.e. $c(X')$ should be as low as possible. Therefore costs of variables should be standardized. Let us denote c_{max} as the maximal cost of all combinations of variables and c_{min} as the minimal cost of all combinations of variables. In the following, we will restrict ourselves to the special case, where the cost of variable combination is defined as sum of costs of variables X_i creating this combination.

$c(X') = \sum c(X_i)$, for all X_i belonging to X'. Then

$$c_{max} = c(X) = \sum_{i=1}^{r} c(X_i)$$

$$c_{min} = min_{i \in \{1,2,\ldots,r\}} c(X_i)$$

Standardized cost of variable combination is defined as

$$c_{STD}(X') = \frac{c(X') - c_{min}}{c_{max} - c_{min}}$$

This equation implies that $c_{STD}(X') \in < 0, 1 >$.

We can define the following criterion function used as criteria of optimality:

$$J(X') = \alpha.\delta(Y \mid X') + (1-\alpha).(1 - c_{STD}(X')), \alpha \in < 0, 1 >$$

The constant α allows us to state the preference of influence $\delta(Y \mid X)$ to standardized cost $c_{STD}(X)$ during the search in selection algorithm. The bigger the constant α is, the more important is the influence, the smaller the α is, the cost becomes more important.

Algorithms for selection of optimal subset of variables

Our problem is to find the optimal or at least sub-optimal combination $X' \subseteq X$ of independent variables, bringing sufficient information to predict set Y of dependent variables. As shown in Cover [12], in order to guarantee the finding of an optimal subset of r' variables from the given r variables, exhaustive search is a necessary procedure. Exhaustive search examines all subsets of size r'. In many practical cases the values of r' and r result in the number of possible subsets that are too large, i.e. the search takes too long, requires much memory, etc. Therefore, some computationally feasible procedures to avoid the exhaustive search are essential even though the variable set obtained may be suboptimal.

The variable search procedures can be classified into two categories, according to the way that possible candidate variable sets are searched. One are the optimal and the other are the suboptimal search procedures. In the next section, two suboptimal search procedures [13] will be described.

Influence-cost forward algorithm

An influence-cost forward algorithm uses a parameter $J_0 \in (0,1)$, that is a priori chosen close to 1. The procedure starts from $X' = \emptyset$ and we proceed in selection of available variables from X in the following way. The i-th step of procedure is described as follows. $X^{i-1} = \{X_1, X_2, ..., X_{i-1}\}$ are already selected variables from X. We search for a variable Z from $X \setminus X^{i-1}$ maximizing $J(X')$, where $X' = (X^{i-1}, Z)$. This variable is denoted as X^i and considered as the next selected relevant variable for decision making task. If $J(X') > J_0$, then the procedure stops and the set of variables $X^i = \{X_1, X_2, ..., X_i\}$ is the result of the influence-cost forward algorithm, i.e. $X' = X^i$. Otherwise we repeat this procedure till all variables are selected.

Influence-cost backward algorithm

An influence-cost backward algorithm starts with all variables from X. Then we proceed in omitting variables in the i-th step as follows. $Z^{i-1} = \{Z_1, Z_2, ..., Z_{i-1}\}$ are already omitted variables from X. We search for such a variable Z_i from $X \setminus Z^{i-1}$ maximizing $J(X')$, where $X' = X \setminus (Z^{i-1}, Z_i)$. This variable is considered as the next omitted variable for a given decision task and $Z^i = \{Z_1, Z_2, ..., Z_i\}$. If $J(X')$ is less or equal to J_0, the procedure stops and the set of variables $X' = X \setminus Z^{i-1}$ is the result of the influence-cost backward algorithm. Otherwise we continue in selection of the next unrelevant variable till all variables are selected.

Possible improvement of this searching techniques can be the combination of above mentioned algorithms, e.g. the floating algorithm. This algorithm combines forward inclusion of variables into set X' and omitting of unrelevant variables using backward algorithm. This approach improves the selection by examining the other combinations and better handles with the nesting effect.

References

[1] Zvrov J., Studeně M., Preiss J. : On extracting relevant information from medical data. MEDINFO 96 proceedings, J. Brender et al., Amsterdam : IOS Press, 1996, 649-653

[2] Zvrov J., Studeně M. : Information theoretical approach to constitution and reduction of medical data. Int J Med Inf 1997 Jun;45(1-2), 65-74

[3] Zvrov J., Tome»kov M., tefek M., Boudk F., Zvra K. : Decision support and data analysis tools for risk assessment in primary preventive study of atherosclerosis. MEDINFO 97 proceedings, C. Pappas et al., Amsterdam : IOS Press, 1997, 625-628

[4] Vajda I. : On the f-divergence and singularity of probability measures. Periodica Math Hung 2, 1972, 223-234

[5] Vajda I. : Theory of statistical inference and information, Kluwer, Dordrecht, 1989

[6] Perez A. : Information-theoretical risk estimates in statistical decision, Kybernetika 3, 1967, 11-21

[7] Perez A. : e-admissible simplifications of the dependence structure of a set of random variables, Kybernetika 13, 1977, 439-449

[8] Studeně M. : Asymptotic behaviour of empirical multiinformation, Kybernetika 23, 1987, 124-135

[9] Studeně M. : Multiinformation and the problem of characterization of conditional independence relations, Probl. control info. theory 18, 1989, 3-16

[10] Zvrov J. : Expert systems and relevant information, Environmetrics 10, 1999, 493-504

[11] Pierik F.H., van Ginneken A.M., Timmers T., Stam H., Weber R.F.: Restructuring routinely collected patient data: ORCA applied to andrology. Yearbook of Medical Informatics 98, Schattauer, Stuttgart 1998

[12] Cover T.M., van Campenhout J.M.: On the possible orderings in the measurement selection problem. IEEE Transactions on System, Man and Cybernetics, SMC-7, 1977, 657-661

[13] Pudil P.: Methodology of Feature Selection in Statisticaal Pattern Recognition, Faculty of Management, Jindýichv Hradec, 1998

Vztah Lanczosovy metody k ostatním Krylovovským metodám

doktorand:
MGR. PETR TICHÝ
ÚI AVČR, Pod vodárenskou věží 2, Praha 8, ČR
petr.tichy@centrum.cz

školitel:
DOC. RNDR. JAN ZÍTKO, CSC.
MFF UK, Sokolovská 83, Praha 2, ČR
zitko@karlin.mff.cuni.cz

obor studia:
M11 – Vědecko-technické výpočty

Abstrakt

Jeden z parametrů metod pro řešení soustavy lineárních rovnic založených na Lanczosově procesu, jež vytváří biortogonální posloupnosti vektorů, je levý startovací vektor, často označovaný jako stínový vektor. Ukážeme volby tohoto vektoru, které způsobí rovnost některých residuí spočtených Lanczosovou metodou s rezidui jiné Krylovovské metody. V pravděpodobnostním smyslu je skoro-vždy možné nalézt stínový vektor vedoucí k rovnosti k-tých residuí. Pokusíme se naznačit, na čem závisí vztah residuí Lanczosovy metody a libovolné Krylovovské metody.

1. Úvod

Uvažujme systém lineárních rovnic

$$\mathbf{A}x = b, \tag{64}$$

kde $\mathbf{A} \in \mathbb{R}^{n\times n}$ je reálná regulární matice, $b \in \mathbb{R}^n$ je vektor pravé strany, $x^* \in \mathbb{R}^n$ přesné řešení soustavy (64). Důležitou třídu metod na řešení soustavy (64) tvoří metody založené na projektivních technikách, hledajících aproximaci řešení soustavy (64) ve vhodném prostoru $\mathcal{K} \in \mathbb{R}^n$ dimenze k. Aby bylo možné tuto aproximaci sestrojit, je nutné předepsat k určujících podmínek. Jedním z možných způsobů volby těchto podmínek jsou ortogonální podmínky, přesněji řečeno, aproximace $x \in \mathcal{K}$ je dána podmínkou kolmosti residuového vektoru $b - Ax$ na k lineárně nezávislých vektorů, jež společně určují prostor $\mathcal{L}$ dimenze k. Tento prostor bývá často nazýván ***levým*** prostorem. Popsané určující podmínky jsou běžně používány v mnoha různých matematických metodách a jsou známy jako ***Petrov-Galerkinovy*** podmínky a při volbě $\mathcal{L} \stackrel{\text{def}}{=} \mathcal{K}$ jako ***Galerkinovy*** podmínky.

Při řešení soustav lineárních rovnic je někdy k dispozici počáteční přiblížení x_0 a je tedy vhodné ho zabudovat do našich úvah například tak, že aproximaci řešení x nebudeme hledat v homogenním prostoru $\mathcal{K}$, ale ve varietě $x_0 + \mathcal{K}$ a opět bude určena podmínkou $b - Ax \perp \mathcal{L}$. V konkrétních realizacích tohoto přístupu vytváříme postupně posloupnosti prostorů $\mathcal{K}_k$ a $\mathcal{L}_k$ a používáme výše popsaný projekční krok. Příkladem volby posloupnosti těchto prostorů je $\mathcal{K}_k = \mathcal{L}_k = \text{span}\{e_1, \ldots, e_k\}$, kdy dostáváme Gaussovu-Seidelovu iterační metodu.

Důležitou třídu metod na řešení soustavy (64) tvoří metody, u nichž se za posloupnost prostorů $\mathcal{K}_k$ volí posloupnost Krylovových prostorů $\mathcal{K}_k(\mathbf{A}, r_0) \stackrel{\text{def}}{=} \text{span}\{r_0, \mathbf{A}r_0, \dots, \mathbf{A}^{k-1}r_0\}$, $r_0 \stackrel{\text{def}}{=} b - \mathbf{A}x_0$, a k-tá aproximace řešení $x_k \in x_0 + \mathcal{K}_k(\mathbf{A}, r_0)$ soustavy (64) je dána pomocí k určujících podmínek aplikovaných na vektor $r_k \stackrel{\text{def}}{=} b - \mathbf{A}x_k$. Tuto třídu metod nazveme "metody Krylovových prostorů", v dalším budeme užívat označení ***Krylovovské metody***. Protože lze tyto podmínky pro residuum převést na podmínku ortogonality residua k určitému prostoru, jsou námi definované Krylovovské metody metodami projekčními a platí pro ně

$$x_k \in x_0 + \mathcal{K}_k(\mathbf{A}, r_0), \quad b - \mathbf{A}x_k \perp \mathcal{L}_k, \tag{65}$$

kde $\mathcal{L}_k$ je vhodně zvolený prostor dimenze k. Poznamenejme, že residuum r_k leží podle (65) ve varietě

$$r_0 + \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0). \tag{66}$$

Z (65) plyne, že rostou-li dimenze obou prostorů, získáme v nejvýše v n-tém kroku přesné řešení x^* soustavy (64). Značnou libovůli máme v určování levého prostoru $\mathcal{L}_k$. Volíme-li $\mathcal{L}_k \stackrel{\text{def}}{=} \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0)$, docílíme vlastnosti

$$\|b - \mathbf{A}x_k\| = \min_{x \in x_0 + \mathcal{K}_k(\mathbf{A}, r_0)} \|b - \mathbf{A}x\|$$

a residuum $r_k = b - Ax_k$ má nejmenší euklidovskou normu ze všech přípustných residuí. Metoda pro počítání těchto residuí a aproximací je známa jako GMRES (Generalized Mimimal Residual Method). Volba $\mathcal{L}_k \stackrel{\text{def}}{=} \mathcal{K}_k(\mathbf{A}, r_0)$ způsobí, že počítaná residua budou vzájemně ortogonální. Metodu počítající uvedené vektory označíme jako FOM (Full Orthogonalization Method). Konečně, volíme-li $\mathcal{L}_k \stackrel{\text{def}}{=} \mathcal{K}_k(\mathbf{A}^T, \widetilde{r}_0)$, kde $\widetilde{r}_0$ je libovolný nenulový vektor zvaný též stínový vektor, získáme Lanczosovu metodu (LM). Upozorněme ještě, že pod pojmem *metoda* rozumíme libovolný algoritmus počítající vektory uvedených vlastností.

Právě díky volbě levého prostoru se svět Krylovovských metod dělí na dvě části. Na metody, které mají jisté "dobré" teoretické vlastnosti (GMRES, FOM) avšak vysoké paměťové nároky (splnění ortogonální podmínky obecně vyžaduje přítomnost všech vektorů báze prostoru $\mathcal{K}_k(\mathbf{A}, r_0)$) a na metody s nízkými paměťovými nároky, postrádající "dobré" teoretické vlastnosti (LM, QMR), avšak v praxi většinou dobře fungující vždy, když fungují metody GMRES a FOM. První třídu metod nazýváme metody s ***dlouhými*** a druhou třídu metody s ***krátkými*** rekurentními vztahy (dále jen ***rekurencemi***).

Zatímco konvergenční křivky GMRES a FOM jsou jednoznačně určeny počáteční aproximací x_0 a maticí $\mathbf{A}$, vystupuje v Lanczosově metodě navíc parametr $\widetilde{r}_0$, pomocí nějž můžeme "hýbat" s konvergenční křivkou této metody a snažit se přiblížit jí ke konvergenčním křivkám ostatních metod. I když tento postup pravděpodobně postrádá praktický význam (spočtení takovéhoto vektoru bude jistě početně velmi náročné), mohl by značně přispět k porozumnění konvergence Lanczosovy metody a jejího vztahu k metodám s dlouhými rekurencemi.

2. Lanczosova metoda a obecná tříkroková metoda[1]

Lanczosova metoda počítá aproximace x_k^L a residuové vektory $r_k^L \stackrel{\text{def}}{=} b - \mathbf{A}x_k^L$ určené podmínkami

$$x_k^L \in x_0 + \mathcal{K}_k(\mathbf{A}, r_0), \quad r_k^L \perp \mathcal{K}_k(\mathbf{A}^T, \widetilde{r}_0). \tag{67}$$

Již Lanczos samotný [3], [4] ukázal, že lze tyto vektory počítat pomocí tříkrokových rekurencí

$$\begin{aligned} r_k^L &= \gamma_k(\mathbf{A}r_{k-1}^L - \alpha_k r_{k-1}^L - \beta_k r_{k-2}^L), \\ x_k^L &= -\gamma_k(r_{k-1}^L + \alpha_k x_{k-1}^L + \beta_k x_{k-2}^L). \end{aligned}$$

Abychom vyhověli podmínce $r_k^L \in r_0 + \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0)$, volíme $\gamma_k = -(\alpha_k + \beta_k)^{-1}$. Chceme-li efektivně počítat koeficienty α_k a β_k, je obecně nutné generovat bázi $\{\widetilde{r}_i\}_{i=0}^{k-1}$ prostoru $\mathcal{K}_k(\mathbf{A}^T, \widetilde{r}_0)$. Volíme-li tuto bázi tak, že $\widetilde{r}_k \perp \mathcal{K}_k(\mathbf{A}, r_0)$, lze ji opět počítat pomocí tříkrokové rekurence tvaru

$$\widetilde{r}_k = \widetilde{\gamma}_k(\mathbf{A}^T\widetilde{r}_{k-1} - \alpha_k\widetilde{r}_{k-1} - \widetilde{\beta}_k\widetilde{r}_{k-2}).$$

[1]Tento odstavec je inspirován články A. Greenbaum [1, 2]

Koeficienty $\widetilde{\gamma}_k$ a $\widetilde{\beta}_k$ je nutné volit tak, aby vyhovovali podmínce $\widetilde{\gamma}_k\widetilde{\beta}_k = \beta_k\gamma_k$. Volme v naší implementaci LM $\widetilde{\gamma}_k \stackrel{\text{def}}{=} \beta_k$ a $\widetilde{\beta}_k \stackrel{\text{def}}{=} \gamma_k$.

Při znalosti vektorů r^L_{k-1}, r^L_{k-2}, $\widetilde{r}_{k-1}$, $\widetilde{r}_{k-2}$ potom můžeme počítat čísla α_k a β_k jako

$$\alpha_k = \frac{(\widetilde{r}_{k-1}, \mathbf{A}r^L_{k-1})}{(\widetilde{r}_{k-1}, r^L_{k-1})}, \quad \beta_k = \frac{(\widetilde{r}_{k-2}, \mathbf{A}r^L_{k-1})}{(\widetilde{r}_{k-2}, r^L_{k-2})}.$$

K ukončení algoritmu dochází, je-li $(\widetilde{r}_i, r^L_i) = 0$ nebo $\alpha_i + \beta_i = 0$ pro nějaké i. Roste-li dimenze Krylovových prostorů až do n a nedojde-li k ukončení algoritmu, dostáváme $r_n \perp \mathcal{K}_n(\mathbf{A}^T, \widetilde{r}_0)$ a $\widetilde{r}_n \perp \mathcal{K}_n(\mathbf{A}, r_0)$ a tudíž $r_n = \mathbf{o} = \widetilde{r}_n$. Zapíšeme-li uvedené rekurence maticově, máme v n-tém kroku

$$\mathbf{AV} = \mathbf{VT}, \quad \mathbf{A}^T\mathbf{W} = \mathbf{W}\widetilde{\mathbf{T}},$$

kde $\mathbf{V} \in \mathbb{R}^{n\times n}$ je matice se sloupci $r_0, \ldots, r_{n-1}$, $\mathbf{W} \in \mathbb{R}^{n\times n}$ se sloupci $\widetilde{r}_0, \ldots, \widetilde{r}_{n-1}$ a $\mathbf{T}$ resp. $\widetilde{\mathbf{T}}$ je třídiagonální matice složená z koeficientů α_k, β_k, γ_k resp. α_k, $\widetilde{\beta}_k$, $\widetilde{\gamma}_k$. Volíme-li, jak již bylo naznačeno v předchozím, $\widetilde{\beta}_k \stackrel{\text{def}}{=} \gamma_k$, platí navíc

$$\widetilde{\mathbf{T}} = \mathbf{T}^T.$$

Uveďme nyní, co rozumíme pod pojmem obecná tříkroková metoda.

Definice 1. *Nechť* $\dim(\mathcal{K}_n(\mathbf{A}, r_0)) = n$. ***Obecnou tříkrokovou metodou*** *rozumíme metodu, počítající residua a aproximace podle předpisu*

$$\begin{aligned} r_k &= \gamma_k(\mathbf{A}r_{k-1} - \alpha_k r_{k-1} - \beta_k r_{k-2}), \\ x_k &= -\gamma_k(r_{k-1} + \alpha_k x_{k-1} + \beta_k x_{k-2}), \end{aligned}$$

kde α_k *a* β_k *jsou libovolné koeficienty,* $\beta_k \neq 0$, $\gamma_k = -(\alpha_k + \beta_k)^{-1}$. *Obecnou tříkrokovou metodu nazveme* n-***finitní****, jestliže nedojde k předčasnému ukončení algoritmu a platí-li* $r_n = \mathbf{o}$.

Věta 1. (***O vztahu mezi Lanczosovou metodou a obecnou tříkrokovou metodou***) *Nechť* $\dim(\mathcal{K}_n(\mathbf{A}, r_0)) = n$. *Potom platí*

1. *Nedojde-li k předčasnému ukončení LM, je LM obecnou n-finitní tříkrokovou rekurencí.*
2. *Pro každou obecnou n-finitní 3-k metodu existuje stínový vektor $\widetilde{r}_0$ takový, že Lanczosova metoda počítá residua a aproximace dané 3-k metody.*
3. *Pro každou obecnou 3-k metodu existuje stínový vektor $\widetilde{r}_0$ takový, že Lanczosova metoda počítá residua a aproximace 3-k metody až do kroku $[n/2]$.*

Důkaz: Bod 1. je důsledkem vlastnosti Lanczosovy metody $r_n \perp \mathcal{K}_n(\mathbf{A}^T, \widetilde{r}_0)$. Je-li obecná 3-k metoda n-finitní, můžeme její rekurence psát v maticovém tvaru $\mathbf{AV} = \mathbf{VT}$. Matice $\mathbf{V}$ je regulární, což to plyne z lineární nezávislosti počítaných vektorů (a ta plyne z rostoucí dimenze Krylovova prostoru). Definujme nyní $\mathbf{W} \stackrel{\text{def}}{=} \mathbf{V}^{-T}$, t.j. platí $\mathbf{W}^T\mathbf{V} = \mathbf{I}$. Transponujeme-li maticovou rovnost $\mathbf{AV} = \mathbf{VT}$ a přenásobíme-li jí maticí $\mathbf{W}$ zprava i zleva, dostáváme

$$\mathbf{A}^T\mathbf{W} = \mathbf{W}\mathbf{T}^T.$$

Zvolme stínový vektor jako první sloupec matice $\mathbf{W}$. Protože jsou touto volbou určeny vektory Lanczosovy metody jednoznačně a vektory matice $\mathbf{W}$ jsou biortogonální k vektorům matice $\mathbf{V}$, počítá zřejmě LM stejné vektory jako daná tříkroková rekurence.

Pro důkaz bodu 3 použijeme vlastnosti skalárního součinu a sice že $((\mathbf{A}^T)^k\widetilde{r}_0, r_k) = (\widetilde{r}_0, \mathbf{A}^k r_k)$. Zvolíme-li $\widetilde{r}_0$ např. jako ortogonální projekci vektoru r_0 na prostor generovaný vektory r_1, r_2, $\mathbf{A}r_2$, …, r_k, $\mathbf{A}r_k$,…,$\mathbf{A}^{k-1}r_k$, splňují vektory r_i podmínky $r_i \perp \mathcal{K}_i(\mathbf{A}^T, \widetilde{r}_0)$ a z jednoznačnosti určení Lanczosových residuí plyne, že $r_i = r^L_i$. Dimenze prostoru, na který má být $\widetilde{r}_0$ kolmý je $2k-1$. Jakmile k přesáhne hodnotu $[n/2]$, je obecně dimenze tohoto prostor rovna n. Pro korektní důkaz tohoto tvrzení bychom museli jít hlouběji do vlastností speciálního prostoru generovaného vektory $r_1, \ldots, r_k$, což učiníme v následujícím odstavci. □

Závěr

Pomocí stínového vektoru můžeme parametrizovat všechny n-finitní obecné tříkrokové metody a dále všechny obecné tříkrokové metody až do kroku $[{}^{n}\!/_{2}]$. Z téhož plyne, že libovolnou 3-k metodu počítající residua r_0, r_1,...,$r_{[n/2]}$ lze dodefinovat tak, aby byla metodou n-finitní.

3. Lanczosova metoda a residua jiné Krylovovské metody

Nechť jsou nyní $r_1, \ldots, r_k$ residua libovolné Krylovovské metody. Definujme prostor

$$\mathcal{W}_k(\mathbf{A}, r_1, \ldots, r_k) \overset{\text{def}}{=} \operatorname{span} \begin{pmatrix} r_1, & r_2, & r_3, & r_4 & \cdots & r_k, \\ & \mathbf{A}r_2, & \mathbf{A}r_3, & \mathbf{A}r_4, & \cdots & \mathbf{A}r_k, \\ & & \mathbf{A}^2 r_3, & \mathbf{A}^2 r_4, & \cdots & \mathbf{A}^2 r_k, \\ & & & \ddots & & \vdots \\ & & & & \ddots & \vdots \\ & & & & & \mathbf{A}^{k-1} r_k \end{pmatrix}.$$

Mohlo by se zdát, že zvolíme-li stínový vektor kolmý na tento prostor, budou residua splňovat $r_i \perp \mathcal{K}_i(\mathbf{A}^T, \widetilde{r}_0)$ a tudíž budou počitatelná Lanczosovou metodou. Ukážeme, proč tomu tak není. Pišme v dalším pouze $\mathcal{W}_k$, budeme-li mít na mysli obecnou množinu residuí $r_1, \ldots, r_k$.

Lemma 1. *Platí buď* $\mathcal{W}_k = \mathcal{K}_{2k}(\mathbf{A}, r_0)$ *nebo je* $\dim(\mathcal{W}_k) = 2k - 1$ *a* $r_0 \notin \mathcal{W}_k$.

Důkaz: Každý vektor z prostoru $\mathcal{W}_k$ leží v $\mathcal{K}_{2k}(\mathbf{A}, r_0)$ a proto je $\mathcal{W}_k \subseteq \mathcal{K}_{2k}(\mathbf{A}, r_0)$. Jelikož lze nalézt $2k-1$ lineárně nezávislých vektorů platí $\dim(\mathcal{W}_k) \geq 2k-1$. Pokud $r_0 \in \mathcal{W}_k$ lze nalézt v $\mathcal{W}_k$ $2k$ lineárně nezávislých vektorů a $\mathcal{W}_k = \mathcal{K}_{2k}(\mathbf{A}, r_0)$. V opačném případě je $\mathcal{W}_k \subset \mathcal{K}_{2k}(\mathbf{A}, r_0)$ a tudíž je $\dim(\mathcal{W}_k) = 2k-1$. □

Lemma 2. *Předpokládejme Krylovovskou metodu jejíž residua jsou počítány podle předpisu*

$$r_i = \alpha_i^{(i)} \mathbf{A} r_{i-1} + \sum_{j=0}^{i-1} \alpha_j^{(i)} r_j, \quad \sum_{j=0}^{i-1} \alpha_j^{(i)} = 1, \quad \alpha_i^{(i)} \neq 0. \tag{68}$$

Potom vektor r_0 *neleží v prostoru* $\mathcal{W}_k$ *tehdy a jen tehdy, je-li rekurence (68)*

(a) *obecná tříkroková*

(b) *tvaru*

$$\begin{aligned} r_1 &= \alpha_1^{(1)} \mathbf{A} r_0 + r_0 \\ r_j &= \alpha_j^{(j)} \mathbf{A} r_{j-1} + \alpha_{j-1}^{(j)} r_{j-1} + \alpha_{j-2}^{(j)} r_{j-2}, \\ &\quad \alpha_{j-2}^{(j)} \neq 0, \ j = 2, \ldots, i-1. \\ r_i &= \alpha_i^{(i)} \mathbf{A} r_{i-1} + r_{i-1} \\ r_s &= \alpha_s^{(s)} \mathbf{A} r_{s-1} + \alpha_{s-1}^{(s)} r_{s-1} + \ldots + \alpha_{i-1}^{(s)} r_{i-1}, \\ &\quad s = i+1, \ldots, k. \end{aligned}$$

Důkaz: Je technického charakteru a neuvádíme ho. □

Lemma 3. *Volme* $\widetilde{r}_0 \perp \mathcal{W}_k$, $k \leq [{}^{n}\!/_{2}]$. *Potom nastane právě jedna z následujících možností*

1. $\dim(\mathcal{W}_k) = 2k \Rightarrow r_0 \in \mathcal{W}_k$ *a dojde k ukončení algoritmu LM v prvním kroku* ($\widetilde{r}_0^T r_0 = 0$).

2. $\dim(\mathcal{W}_k) = 2k-1$ *a rekurence počítající residua jsou tvaru (b). Potom dojde k ukončení algoritmu LM v kroku i a platí $\widetilde{r}_i^T r_i = 0$.*

3. $\dim(\mathcal{W}_k) = 2k-1$ *a rekurence jsou tvaru (a). V tomto případě spočte LM residua $r_1, \ldots, r_k$.*

Důkaz: Důkaz prvního tvrzení je zřejmý. Prokázání zbylých dvou je technického charakteru. □

Vidíme, že všechna residua $r_1, \ldots, r_k$ nějaké Krylovovské metody lze počítat LM právě tehdy, jsou-li počitatelná obecnou 3-k metodou. Vektor $\widetilde{r}_0$ však nemusíme volit kolmý na celý prostor $\mathcal{W}_k$, ale pouze na nějaký jeho podprostor, splňující nutné podmínky, že neobsahuje vektor r_0 a jeho dimenze je menší než $2k$.

Věta 2. (***Lanczosova metoda a residua jiné Krylovovské metody***)

- *Nechť r_k je residuum počítané obecnou rekurencí (68). Zvolme stínový vektor tak, že $(\widetilde{r}_0, r_0) \neq 0$ a $\widetilde{r}_0 \perp \mathcal{K}_k(\mathbf{A}, r_k)$. Nedojde-li k předčasnému ukončení algoritmu LM, je $r_k^L = r_k$.*
- *Nechť r_i jsou residua počítaná obecnou rekurencí (68), $i = 1, 2, 4, 8, \ldots, 2^l$, $2^l \leq [n/2]$. Definujme prostor*
$$\mathcal{Z}_{2^l} \overset{def}{=} \bigcup_{i=0}^{l} \mathcal{K}_{2^i}(\mathbf{A}, r_{2^i}),$$
a volme stínový vektor tak, že je ortogonální k tomuto prostoru, $\widetilde{r}_0 \perp \mathcal{Z}_{2^l}$, $(\widetilde{r}_0, r_0) \neq 0$. Nedojde-li k předčasnému ukončení algoritmu LM, je $r_1^L = r_1$, $r_2^L = r_2$, $r_4^L = r_4$, $\ldots$, $r_{2^l}^L = r_{2^l}$.

Důkaz: plyne z existence a jednoznačnosti určení residuí LM. □

Věta 3. (***Lanczosova metoda a k-té residuum***) *Nechť r_k je residuum počítané obecnou rekurencí (68). Definujme prostory $\mathcal{R}_k \overset{def}{=} \mathcal{K}_k(\mathbf{A}, r_k)$ a $\mathcal{H}_k \overset{def}{=} \mathcal{R}_k^{\perp}$. Potom platí jedno z následujících dvou tvrzení*

- *Lanczosova metoda spočte residuum r_k pro skoro všechny stínové vektory $\widetilde{r}_0 \in \mathcal{H}_k$.*
- *Pro každý vektor $\widetilde{r}_0 \in \mathcal{H}_k$ dojde k předčasnému ukončení v kroku $i < k$.*

Důkaz: Provedeme v podobném stylu jako jsou dokazována tvrzení v [5]. K předčasnému ukončení LM dochází tehdy, je-li $(\widetilde{r}_i, r_i) = 0$ pro nějaké i. Volíme-li $\widetilde{r}_0 \in \mathcal{H}_k$, můžeme ho vyjádřit ve tvaru
$$\widetilde{r}_0 = \sum_{i=1}^{n-k} \xi_i h_i,$$
kde h_i tvoří bázi prostoru $\mathcal{H}_k$. Potom skalární součiny $(\widetilde{r}_i, r_i)$ nejsou nic jiného než racionální funkce proměnných $\xi_1, \ldots, \xi_{n-k}$. Tyto racionální funkce jsou buď netriviální a tedy skoro-všude nenulové a nebo identicky rovny nule a pak dojde k ukončení v i-tém kroku pro každý vektor $\widetilde{r}_0 \in \mathcal{H}_k$. □

Příklad:
$$(r_0, \widetilde{r}_0) = \sum_{i=1}^{n-k} \xi_i (r_0, h_i),$$
je dle definice prostoru $\mathcal{H}_k$ a faktu $r_0 \notin \mathcal{R}_k$ netriviální lineární funkcí proměnných ξ_i. Dále
$$(\widetilde{r}_1, r_1) = \gamma_1 \widetilde{\gamma}_1 \frac{(\widetilde{r}_0, \mathbf{A}^2 r_0)(\widetilde{r}_0, r_0) - (\widetilde{r}_0, \mathbf{A} r_0)^2}{(\widetilde{r}_0, r_0)}$$

je dobře definovaná racionální funkce, která však může být identicky rovna nule například tehdy, jsou-li si projekce vektorů $r_0, \mathbf{A}r_0, \mathbf{A}^2 r_0$ na prostor $\mathcal{H}_k$ rovny. Každý vektor $v \in \mathbb{R}^n$ můžeme totiž psát ve tvaru

$$v = y + z, \ y \in \mathcal{H}_k, \ z \in \mathcal{R}_k, \ (y, z) = 0,$$

kde y je projekce na prostor $\mathcal{H}_k$ a z na prostor $\mathcal{R}_k$. Označíme-li $\mathbf{A}^i r_0 = y_i + z_i$, $y_i \in \mathcal{H}_k$, $z_i \in \mathcal{R}_k$ potom je

$$(\widetilde{r}_0, \mathbf{A}^i r_0) = (\widetilde{r}_0, z_i + y_i) = (\widetilde{r}_0, y_i).$$

Platí-li tedy $y_0 = y_1 = y_2$, je $(\widetilde{r}_1, r_1) = 0$ pro každý $\widetilde{r}_0 \in \mathcal{H}_k$. □

Závěr

Může nastat situace, kdy k-té residuum nějaké Krylovovské metody není počitatelné Lanczosovou metodou. Jak je vidět z příkladu, tato situace nastává při velmi speciální volbě počátečního residua, matice $\mathbf{A}$ a k-tého residua z variety $r_0 + \mathbf{A}\mathcal{K}_k(\mathbf{A}, r_0)$ jež chceme počítat.

4. Vztah Lanczosovy metody a obecné Krylovovské metody z hlediska residuí

Ve své práci [2] Anne Greenbaum ukazuje, že existuje dvoukroková metoda, která je velmi blízká optimální GMRES, konkrétněji, residua této dvoukrokové rekurence r_k a residua metody GMRES splňují nerovnost

$$\|r_k\| \le \sqrt{(k+1)(n-k)}\kappa(\mathbf{Z})\|r_k^G\|,$$

kde $\kappa(\mathbf{Z})$ je číslo podmíněnosti nejlépe podmíněné matice vlastních vektorů matice $\mathbf{A}$. Protože je třída dvoukrokových metod pouze podmnožinou tříkrokových, lze tušit existenci "optimálního" stínového vektoru, který bude zajišťovat blízkost Lanczosovy metody k danné Krylovovské metodě. Není zatím jasné jak tento vektor zkonstruovat, zřejmě jeden z podstatných vlivů bude poloha stínového vektoru a každého z prostorů $\mathcal{K}_k(\mathbf{A}, r_k)$. Dalším nástrojem k poodhalení problému stínového vektoru by mohly být numerické experimenty, zkoumající "citlivost" při jednotlivých reziduích metody, k níž se chceme přiblížit, konktrétněji, k danému residuu r_k umíme sestrojit lokálně optimální stínový vektor $\widetilde{r}_0^{LO}$ takový, že LM počítá residuum r_k. Nyní můžeme zkoušet volit

$$\widetilde{r}_0 = \widetilde{r}_0^{LO} + \sum_{i=1}^{k} \epsilon_i m_i,$$

kde m_i je ortonormální báze $\mathcal{K}_k(\mathbf{A}, r_k)$ a sledovat změny normy residua, mění-li se stínový vektor v různých směrech o různé hodnoty a doufat, že dostaneme odpovět na podstatné otázky pro praktické vypočty. Generujeme-li totiž stínový vektor náhodně, dostáváme velmi podobné křivky, které mají stejné globální chování, ačkoliv teorie říká, že můžeme očekávat téměr cokoliv. Podstatné otázky tedy zní: Proč jsou si podobné konvergenční křivky a na čem závisí toto globální chování?

References

[1] A. Greenbaum: "On the role of the left starting vector in the two-sided Lanczos algorithm and nonsymetric linear system solvers". *Proceedings of the Dundee meeting*, in Numerical Analysis 1997, D.F. Griffiths, D.J. Highham, G.A. Watson, eds., Pitman Research Notes in Mathematics Series 380, Longman, 1997.

[2] A. Greenbaum: "Relation of a Two-Term Recurrence for Nonhermitian Linear Systems to GMRES", 1999.

[3] C. Lanczos: "An iteration method for the solution of eigenvalue problem of linear differential and integral operators", *J. Res. Nat. Bureau Standards 45*, 1950.

[4] C. Lanczos: "Solution of systems of linear equations by minimized iterations", *J. Res. Nat. Bureau Standards 49*, 1952.

[5] W.D. Joubert: "Lanczos methods for the solution of nonsymmetric systems of linear equations", *SIAM J. Matrix Anal. Appl. 13*, 926-943, 1992.

Algorithms for Decomposable BSP Computers

doktorand:
Martin Beran

SISAL MFF UK, Malostranské nám. 25, 118 00 Praha 1

beran@ss1000.ms.mff.cuni.cz

školitel:
Doc. RNDr. Jiří Wiedermann, DrSc.

ÚI AV ČR, Pod vodárenskou věží 2, 182 07 Praha 8

wieder@cs.cas.cz

obor studia:
I1 — Teoretická informatika

Abstrakt

The bulk-synchronous parallel (BSP) computer — introduced by Valiant in [1] — is a paradigmatic example of a bridging model of parallel computation. It is a parametrized model that realistically describes performance of existing parallel computers while retaining independence on concrete architectures. We define an extended model: decomposable BSP (dBSP). To illustrate the power of dBSP, we will show how some elementary BSP algorithms (broadcasting, prefix sums, matrix multiplication, and simulation of cellular automata) are sped up by their adaptation to the decomposable BSP model.

1. Introduction

The bulk-synchronous parallel (BSP) computer [1] is a widely accepted model of parallel computation. An extensive research on BSP algorithms and implementation of the model on real computers has been done in recent years [2, 3, 4, 5, 6, 7].

The standard BSP charges communication between any pair of processors equally. Thus, it cannot exploit communication locality present in many algorithms. By communication locality we mean that a processor communicates not with all, but only with "close" (in some sense) other processors. Not distinguishing between communication to "short" and "long" distances yields too pessimistic estimates of time complexity in some cases.

The BSP model can be extended with ability to exploit locality. One possible extension is the decomposable BSP (dBSP) defined in [8]. We will describe how some simple BSP algorithms can be adapted for dBSP. For each algorithm, we will compare execution times on both models and show that dBSP provides a significant speedup.

2. BSP and dBSP Models

The *Bulk Synchronous Parallel (BSP) Computer* consists of p processors with local memories. The processors can communicate by sending messages via a router. The computation runs in supersteps, i.e., the processors work asynchronously, but are periodically synchronized by a barrier. A superstep consists of three phases: computation, communication, and synchronization. In the computation phase, the processors compute with locally held data. The communication phase consists of a realization of so-called h-relation, i.e., processors send point-to-point messages to other processors so that no processor sends nor receives more than h messages. The data sent in one superstep are available at their destinations from the beginning of the next superstep. In the final phase of each superstep, all the processors perform a barrier synchronization. The performance of the router is given by two parameters g (the ratio of the time needed to send or receive one message to the time of one elementary computational operation — the inverse of the communication throughput) and l (the communication latency and the synchronization overhead). If a BSP computation consists of s supersteps and the i-th superstep is composed of the w_i computational steps in every processor and of the h_i-relation, then the time complexity of the computation is defined

$$T^{\mathrm{BSP}} = \sum_{i=1}^{s}(w_i + h_i g + l) = W + Hg + sl \ ,$$

where $W = \sum_{i=1}^{s} w_i$ and $H = \sum_{i=1}^{s} h_i$.

If the BSP model is realized by some real computer, the complexity of communication typically increases with the number of processors p. Therefore the parameters g and l are not constants, but nondecreasing functions of p, i.e., $g = g(p)$ and $l = l(p)$.

The *Decomposable Bulk Synchronous Parallel (dBSP) Computer* is an extension of the BSP model. An algorithm for dBSP works exactly in the same way as a BSP algorithm except that it can state explicitly that no communication is performed among some processors during a part of computation. Two new instructions `split` and `join` are introduced. After a `split`, the processors of the dBSP machine are partitioned (decomposed) into clusters. No communication is allowed between processors from different clusters until the partitioning is cancelled by `join`. Processors within a cluster can communicate freely. Splitting and joing occurs as a part of synchronization at the end of a superstep. Partitioning can be recursive (splitting clusters into sub-clusters). Repartitioning into different clusters is also possible, but only by a `join` followed by another `split`. A processor cannot be directly moved from one cluster to another. The values of parameters g and l depend on the size of a cluster, thus smaller clusters yield faster communication. The time complexity of a dBSP computation is

$$T^{\mathrm{dBSP}} = \sum_{i=1}^{s}(w_i + h_i g(p_i) + l(p_i)) = W + \sum_{i=1}^{s}(h_i g(p_i) + l(p_i)) \ ,$$

where p_i is the size (number of processors) of the largest non-partitioned cluster existing in the superstep s.

3. Algorithms for dBSP Computers

For each presented algorithm, BSP and dBSP time complexities are compared. General formulas determining the time complexity and the speed-up factor (in comparison with the BSP) are shown as well as results for concrete functions $g(p) = l(p) = \Theta(p^a)$ for some constant $0 < a \leq 1$. These values correspond to an implementation of the BSP model by a mesh network of the dimension $1/a$.

3.1. Broadcasting

A value x stored in the processor 0 is to be transferred to all processors. Classical parallel broadcasting algorithm [5] uses communication structured as a binary tree. Assume that the number of processors is $p = 2^n$. The algorithm has $\log p$ phases (supersteps). In the first superstep, the value x is sent from the processor 0 to the processor $p/2$. In the second superstep, the processor 0 sends to the processor $p/4$ and $p/2$ to $3p/4$. Generally, in every superstep, each processor already possessing x sends it to another processor. After $\log p$ supersteps, x is distributed to all the processors. The BSP time complexity of the algorithm is

$$T^{\mathrm{BSP}}(p) = \Theta\Big((g(p) + l(p))\log p + l(p)\Big) = \Theta(p^a \log p) \ .$$

All communication between the first and the second half of processors is done in the first superstep. Then the halves of the computer work independently for the rest of the computation. Thus a dBSP machine can split itself into 2 clusters of $p/2$ processors after the first superstep. This halving can be done recursively and after s supersteps, the computer will be partitioned into 2^s clusters of $p/2^s$ processors. The corresponding execution time is then

$$T^{\mathrm{dBSP}}(p) = \Theta\left(\sum_{i=1}^{\log p}(g(2^i) + l(2^i)) + l(1)\right) == \Theta\left(\sum_{i=1}^{\log p} 2^{ai+1}\right) = \Theta(p^a) \ .$$

The speedup factor achieved by the dBSP computer in comparison with the standard BSP is $T^{\mathrm{BSP}}(p)/T^{\mathrm{dBSP}}(p) = \log p$.

The broadcasting algorithm can be generalized to a k-ary tree, i.e., a processor sends the value x to $k-1$ other processors in each superstep. There are several algorithms which use the same communication scheme as broadcasting, e.g., aggregation and computation of prefix sums.

3.2. Broadcasting of n Elements

This task is similar to the previous one, but now an array $[x_0, \ldots, x_{n-1}]$ on n values has to be broadcasted. There is an optimal BSP algorithm for the n-element broadcast running in three supersteps [5]. In the first superstep, the processor 0 splits the array X into p chunks of n/p elements and sends each chunk to a different processor. In the second superstep, each processor sends a copy of its chunk to every other processor. Every processor receives all chunks in the third superstep. The algorithm runs in time

$$T^{\mathrm{BSP}}(n,p) = \Theta(ng(p) + l(p)) = \Theta(np^a) \ .$$

This algorithm uses an all-to-all communication pattern. Hence the communication cannot be easily partitioned into disjoint clusters. Adaptation of the BSP algorithm for the dBSP model uses repartitioning of clusters. The machine is first partitioned into $\sqrt{p}$ clusters of $\sqrt{p}$ processors. The BSP algorithm is run in the cluster C containing the processor 0. Then repartitioning into different $\sqrt{p}$ clusters is performed so that each of the new clusters contains exactly one processor from C. Finally, The BSP algorithm is run separately on every cluster. The algorithm time complexity is composed of the cost of repartitioning and of the BSP n-element broadcasting on a $\sqrt{p}$-processor machine.

$$T^{\mathrm{dBSP}}(n,p) = \Theta\Big(2l(p) + 2T^{\mathrm{BSP}}(n, \sqrt{p})\Big) = \Theta\Big(ng(\sqrt{p}) + l(p)\Big) = \Theta(np^{a/2} + p^a) \ .$$

The speedup is $T^{\mathrm{BSP}}(p)/T^{\mathrm{dBSP}}(p) = p^{a/2}$, assumed $n = \Omega(p^{a/2})$. Again, a similarly structured algorithm can be used also for aggregation.

3.3. Dense matrix multiplication

We describe an algorithm for multiplication of 2 matrices with $n \times n$ elements [7]. The matrices A and B are both partitioned into $\sqrt{p} \times \sqrt{p}$ equally sized blocks. Hence, the number of processors used during the computation is $p \leq n^2$. The processor $p_{i,j}$ holds the blocks $A_{i,j}$, $B_{i,j}$ and computes the block $C_{i,j}$ of the result. The BSP computation runs in 2 supersteps. First, every processor $p_{i,j}$ sends $A_{i,j}$ to all $p_{i,k}$ and $B_{i,j}$ to $p_{k,j}$, for all $k \in \{0, \ldots, \sqrt{p}-1\}$. Using the received blocks, $p_{i,j}$ computes $C_{i,j} = \sum_k A_{i,k}B_{k,j}$ in the second superstep. The dBSP modification of this algorithm has 5 supersteps. In the first one, the machine is partitioned so that every row of $\sqrt{p}$ processors belongs to a separate cluster. The second superstep includes exchanging of A blocks in rows and the join operation. Then, a similar partitioning into columns and distributing B blocks is performed in the next 2 supersteps. Finally, blocks of C are computed. The time complexity for $p = n^b$, $0 < b \leq 2$ is

$$\begin{aligned} T^{\mathrm{BSP}}(n,p) &= \Theta\left(\frac{n^3}{p} + \frac{n^2}{\sqrt{p}}g(p) + l(p)\right) = \Theta\left(n^{3-b} + n^{2+ab-b/2}\right) \ , \\ T^{\mathrm{dBSP}(n,p)} &= \Theta\left(\frac{n^3}{p} + \frac{n^2}{\sqrt{p}}g(\sqrt{p}) + l(p) + l(\sqrt{p})\right) = \Theta\left(n^{3-b} + n^{2+ab/2-b/2}\right) \ . \end{aligned}$$

The speedup values according to relation between parameters a and b are summarized in the following table:

	$0 < b \le 2/(2a+1)$	$2/(2a+1) < b < 2/(a+1)$	$2/(a+1) \le b \le 2$
$T^{\mathrm{BSP}}(n)$	$\Theta(n^{3-b})$	$\Theta(n^{2+ab-b/2})$	$\Theta(n^{2+ab-b/2})$
$T^{\mathrm{dBSP}}(n)$	$\Theta(n^{3-b})$	$\Theta(n^{3-b})$	$\Theta(n^{2+ab/2-b/2})$
$T^{\mathrm{BSP}}(n)/T^{\mathrm{dBSP}}(n)$	$\Theta(1)$	$\Theta(n^{ab+b/2-1}) > \omega(1)$	$\Theta(n^{ab/2})$

3.4. Simulation of Cellular Automata

We will restrict ourselves to simulation of 1-dimensional cellular automata, although CA's of any dimension exist. The RAM simulation algorithm straightforwardly initializes an array representing the states of individual cells of the automaton and periodically updates it according to the transition function. The parallel algorithm is based on the RAM algorithm and its structure is the same as in the well known finite difference algorithm [9]. Each processor is assigned a subset of cells. Information about states of cells is periodically exchanged among processors. The automaton is partitioned into blocks of n/p cells, where the number of processors is $p \le n$. Every processors is responsible for processing of one block. In each simulation cycle, k steps of the CA is simulated. To evaluate $q_i^{(t)}$, one must know $q_{i-1}^{(t-1)}$, $q_i^{(t-1)}$, and $q_{i+1}^{(t-1)}$. To compute these three values, $q_{i-2}^{(t-2)}, \dots, q_{i+2}^{(t-2)}$ is needed, and so on. If a processor wants to perform k steps, i.e., to find $q_i^{(t+k)}, \dots, q_{i+n/p-1}^{(t+k)}$, it has to get the values $q_{i-k}^{(t)}, \dots, q_{i+n/p-1+k}^{(t)}$. Consequently, in every simulation cycle, each processor receives $2k$ values from other processors. To make the communication pattern simpler, we require $k \le n/p$. Then messages are sent only between processors holding neighbouring blocks. The communication is performed in two phases. During the first one, the dBSP machine is partitioned into clusters of c processors and data are exchanged among processors belonging to the same cluster. The second phase consists of repartitioning and performing the rest of communication between processors, which were in different clusters during the first phase. Note that simulation of more than 1 step in one cycle induces some redundant computation, because the values of $q_{in/p-k+1}, \dots, q_{in/p+k-2}$ are computed twice (by two neighbouring processors) for $i \in \{1, \dots, p-1\}$. The computational part of every simulation cycle takes time

$$W(n,k) = \Theta\left(\frac{kn}{p} + 2\sum_{i=1}^{k-1} i\right) = \Theta\left(\frac{kn}{p} + k^2\right) \quad .$$

The term $\Theta(k^2)$ corresponds to the redundant computation. The communication consists of one (BSP) or two (dBSP) $2k$-relations. The whole simulation cycle (simulation of k steps) consists of a constant number of supersteps. In total, we get the average time of a single CA step:

$$\begin{aligned} T^{\mathrm{BSP}}(n,p,k) &= \frac{1}{k}\Theta(W(n,k) + 2kg(p) + l(p)) == \Theta\left(\frac{n}{p} + k + g(p) + \frac{l(p)}{k}\right) \quad , \\ T^{\mathrm{dBSP}}(n,p,k,c) &= \frac{1}{k}\Theta(W(n,k) + 4kg(c) + 2l(p) + 2l(c)) == \Theta\left(\frac{n}{p} + k + g(c) + \frac{l(p)}{k}\right) \quad . \end{aligned}$$

For $g(p) = l(p) = p^a$ and with optimal values of other parameters, the optimal execution time is

$$\begin{aligned} T^{\mathrm{BSP}}(n) &= \Theta(n^{a/(a+1)}) \quad , \\ T^{\mathrm{dBSP}}(n) &= \Theta(n^{a/(a+2)}) \quad . \end{aligned}$$

This yields the speedup $T^{\mathrm{BSP}}(n)/T^{\mathrm{dBSP}}(n) = \Theta\left(\frac{n^{a/(a+1)}}{n^{a/(a+2)}}\right) = \Theta\left(n^{\frac{a}{(a+1)(a+2)}}\right)$.

4. Conclusion

We have shown how the dBSP model can be used to improve time complexity of some elementary BSP algorithms. All the presented algorithms possess a regular communication structure and do not use general all-to-all communication patterns. Thus it is possible to perform an efficient partitioning of the decomposable BSP machine.

More detailed description of the algorithms can be found in [10], together with formal definitions of BSP and dBSP, mutual simulations between BSP and dBSP, and results about their relation to other sequential and parallel models of computation.

References

[1] L. G. Valiant, “A bridging model for parallel computation,” *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[2] A. V. Gerbessiotis and C. J. Siniolakis, “Primitive operations on the BSP model,” Tech. Rep. PRG-TR-23-96, Oxford University Computing Laboratory, Oxford, Oct. 1996.

[3] A. V. Gerbessiotis and L. G. Valiant, “Direct bulk-synchronous parallel algorithms,” *Journal of Parallel and Distributed Computing*, vol. 22, pp. 251–267, 1994.

[4] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas, “Towards efficiency and portability: Programming with the BSP model,” in *SPAA '96: Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 1–12, ACM Press, 1996.

[5] B. H. H. Juurlink and H. A. G. Wijshoff, “Communication primitives for BSP computers,” *Information Processing Letters*, vol. 58, pp. 303–310, 1996.

[6] W. F. McColl, “Bulk synchronous parallel computing,” in *Abstract Machine Models for Highly Parallel Computers* (J. R. Davy and P. M. Dew, eds.), pp. 41–63, Oxford University Press, 1995.

[7] W. F. McColl, “Scalable computing,” *Lecture Notes in Computer Science*, vol. 1000, pp. 46–61, 1995.

[8] M. Beran, “Decomposable bulk synchronous parallel computers,” in *Proceedings of SOFSEM '99*, vol. 1725 of *Lecture Notes in Computer Science*, pp. 349–359, Springer-Verlag, 1999.

[9] I. Foster, *Designing and Building Parallel Programs.* Addison-Wesley, 1995.

[10] M. Beran, *Formalizing, Analyzing, and Extending the Model of Bulk Synchronous Parallel Computer.* PhD thesis. In preparation.