ATG Press Appetizer

Volume 1, Number 1, December 2000

Important Note: The information, which is provided by this document, is devoted to the entire world for improving creativity, and therefore developing new job opportunities around the globe, in the field of computer science and engineering. So, no part of the proposed information is protected by any intellectual property right. However, the document itself, is copyrighted by Aria Technology Group. No part of this document could be published, in any form of media, without written permission of Aria Technology Group. This is an edited version of the December 2000 original version. The release date of this issue is December 2004.

Globe CPU

Farnad Laleh
Aria Technology Group

ABSTRACT

Current microprocessor architecture suffers from employing some invented traditions in the 60s. Adhering those traditions has originated many unjustifiable design-level complexities till now. Many approaches that are based on those traditions have come to dead end in recent years. Among them, pipelining and superscalar techniques are notable. In this paper, we show that a powerful microprocessor does not need pipelining and superscalar techniques. The natural result of such an approach, which we call it *globe scale architecture* is the removal of branch prediction, speculative execution, and very long instruction words. Finally, we conclude that a typical CPU based on the proposed architecture dose not need more than the one-tenth of the transistors placed in a typical year-2000 microprocessor. In addition, it is shown that the proposed architecture achieves the best running speed for a sequential computer program.

1. INTRODUCTION

The Beginning of the 1960s was the start of inventing some new methods in computer architecture. Major inventions were made in IBM's STRETCH project. The base of employed methods in today's CPU architecture returns to that time and mainly the IBM 360/91 architecture. Pipelining, out-of-order execution and imprecise interrupt are among them. For that time transistor-based computers, those methods granted high boost to the computation performance. The performance gain motivated current microprocessor architects to employ the same methods to raise their designs performance. However, it was a misleading. The proposed methods had been designed for the technologies of the 60s, and employing them at the beginning of the 90s, which had enabled manufactures to place multi-million transistors on a single chip, was a real misleading. The negative impact of this misleading in today's CPUs is quite evident. Limiting the clock speed for a bug-free pipelining, and an ALU that operates in a double clock rate of the CPU core, are among the results of the misleading. In fact, the methods that were the heroes of the 60s architectures, are the devils of today's.

In this paper, we establish a new computer architecture approach based on today's available resources, rather than the traditions invented by pioneers in the 50s and 60s. In section 2, we discuss about the available resources for computer architects in the 21st century. In section 3, we present the proposed approach. Finally, in section 4, we draw a roadmap for CPU developers in the 21st century.

2. RESOURCES FOR CPU ARCHITECTS

Generally, we have two types of resources for CPU architects: physical and logical. Physical resources derive from advantages and limitations of the physics, e.g., 0.18-micron IC process, and high bandwidth of buses. Logical resources derive from the advantages and constraints of the mathematical rules and theorems in computer science, e.g., the Church-Turing thesis. In this paper, we do not use any new resource type and do not quit from the traditional computer science framework. We just organize the proposed resources in a new fashion.

The number of transistors we can settle on a single die is proportional to the number of functional units on a chip (CPU). However, it is not necessarily proportional to the computational power of a chip. It depends on the size of the transistors and the selected placement and routing strategies for their alignment on the die. Today, we can place up to 50 million transistors on a single die. We do not attempt to do it; rather we want to use the die space for an extended routing strategy. This is our first resource.

The second resource is several buses of 64-bit width or more. We can attach two or more instructions to each other and load them once to the CPU via the proposed buses. It is not for the purposes like superscalar computing or speculative execution.

Conventional CPUs use some functional ALUs to perform their calculating operations. These ALUs are functional units, i.e., their current operation is a function of the executing CPU's instruction. Logically, it is possible to build some ALUs that are not functional and always perform a few fixed operations. This type of ALU that is called *fixed arithmetic logic unit* (FALU) is our third resource (a logical resource).

In the next section, we explain how we can organize these three resources to establish our proposed CPU architecture. This type of architecture is applicable to any type of CPU, e.g., a DSP. In this paper, we focus on microprocessors, and by the term "CPU," we mean a typical general-purpose microprocessor.

3. GLOBE SCALE ARCHITECTURE

In traditional computer science, a CPU or *random access machine*, is a device with a set of instructions that are supposed to be arranged in a desired form for making a desired program. In a regular CPU (in this paper, we call RISC CPUs as regular CPUs because their reduced instruction set is simple for comparison purpose. However, what we explain in this paper, is extensible to CISC and EPIC CPUs as well) we have three major groups of instructions as the following:

1. Data transfer
2. Control
3. Arithmetic and logic

What we call computation is the third group of instructions and the first and the second groups are for arranging the third group in a desired manner. So we have a concept named *arrangement* that employs the first and the second groups. We want to replace the proposed concept with a simple and new concept, which we call it *selection.* The new concept eliminates the third group of instructions and relaxes the first and second groups. To introduce the new concept, consider the following simple example. Suppose you go to a restaurant and see the list of available foods in the menu and order to the servant (CPU) your desired appetizer, meal, and desserts. The servant goes and prepares your proposed foods (arithmetic results) in the order you have requested him. In fact, you *arrange* the works (arithmetic operations) he should do. Now consider you go to a fast food restaurant, which has prepared all of the listed foods before your arrival and demand. What you need to do is to *select* your desired foods (arithmetic results). This is the concept of selection in a CPU, that is, preparing all possible arithmetic results before the program demand, and selecting the proper one on the program demand.

To implement the proposed concept, we replace the so-called *general-purpose registers* with a new type of registers called *wild registers.* The only possible operation that a typical computer program can do with these registers is to write (store) on them. In fact, there is no need to use any other operation for them. They are connected to several FALUs in a combinatorial fashion. This is for producing all possible arithmetic results in parallel. Consequently, the outputs of the FALUs are stored in a new type of registers called *mass registers.* For a computer program, these are read-only registers. Only the FALUs can write on them. The following example shows a better view of the proposed concept.

Suppose we have four wild registers. Similar to conventional CPUs, our arithmetic operations have one or two operand(s). Therefore, the number of possible combinations is $C(4, 2) = 6$ and we need six FALUs. As mentioned before, FALU is not a functional unit and it performs all of its defined operations on its two inputs, in parallel. For a regular CPU, the defined operations are add, subtract, multiply, divide, and, or, xor, shift logical L/R, shift arithmetic, increment, decrement, and compare.

Therefore, every FALU has twenty output lines and the output of each line is stored on a unique mass register. Consequently in a CPU with four integer wild registers, we need $6 \times 20 = 120$ integer mass registers. The same stuff could be considered for floating point wild registers. In a floating point FALU, we do not have logical operations; instead, we have single and double precision operations. The numbers of mass registers for the integer and floating point operations are almost equal.

Wild registers can feed from both of memory and mass registers. Mass registers can send their data to both of memory and wild registers. In the proposed architecture, when we load our desired data into wild registers, all of possible results are prepared in mass registers. Only by selecting the corresponding mass register, we reach our required result. This is the replacement of arrangement by selection. As can be seen, the third group of instructions has been eliminated. Figure 1, shows the concept of wild and mass registers, schematically.

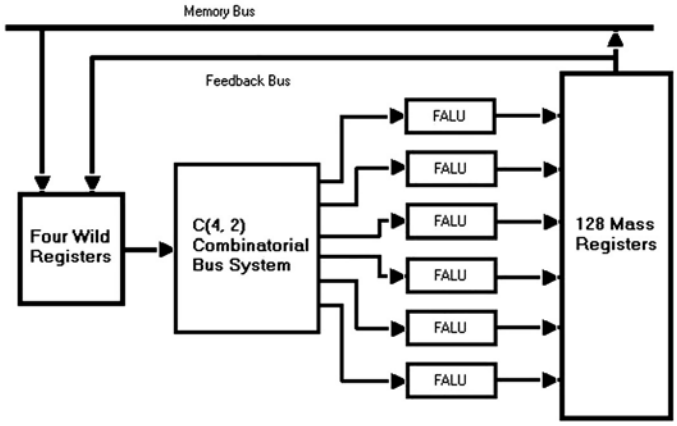

Fig. 1. *Wild and mass registers interconnections*

Currently, we have only two groups of instructions: data transfer and control. Since we use multiple FALUs and each FALU performs several arithmetic operations in parallel, we need an extended number of flag registers. In addition, since the required clock cycles for different arithmetic operations (especially for floating point operations) are different, we need some condition registers to indicate the situations of the performing and terminated arithmetic operations. Since in control instructions, we load/store an address to/from memory pointer registers, we can consider them as data transfer instructions. The only difference between them is the conditional operation. Adding a condition bit to the instruction word, see figure 2, solves the problem. This method is similar to the Predication in EIPC architecture. When the condition bit is on, the instruction is performed conditionally. Using this method, we can unify control and data transfer instructions. Currently, we have only one group of instructions, which is called data transfer.

Let's have a look to the advantages of the elimination of arithmetic instructions and the unification of control and data transfer instructions.

1. Since we have only one type of instructions, we can fix them on a 32-bit instruction word in a decoded mode. So we do not need any decoding process in the CPU cycles.
2. We use combinatorial buses and FALUs. Therefore, we reach the maximum logically available parallelism. In addition, we do not have a separate execution cycle as we have in conventional CPUs. The proposed CPU fetches and executes, independently.

3. Looking to the above advantages, one can find that there is no need for pipelining and therefore branch-prediction and speculative execution. In addition, employing the FALU enables us to achieve maximum parallelism and to dismiss superscalar computing and instruction level parallelism. What we need are high memory bandwidth and low memory latency to load more data to wild registers and read more from mass registers.
4. The proposed CPU has no decode and dynamic execution units. It uses fixed length instructions. Therefore, it employs a reduced number of transistors down to the one-tenth of today's CPUs. However, because of employing combinatorial bus system, it uses an extended on-chip routing. This type of architecture is attractive for multiprocessing on a single chip.
5. With a 32-bit instruction word, we are able to take over most functions. There is no need to use a *very long instruction word.*

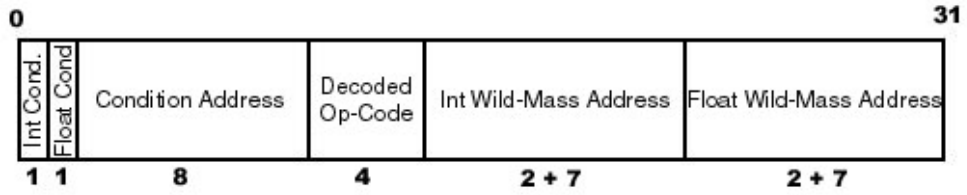

Fig. 2. *A typical GSA instruction word*

Figure 2, shows a typical globe scale architecture (GSA) instruction word in a CPU with four wild registers for each of the integer and floating point operations. The first bit of the word is *integer condition bit.* Whenever this bit is on, the CPU goes to the address of the indicated condition register which is designated by an 8-bit condition address. It is to see if the proposed condition is true or not. If true, the CPU executes instruction. If integer condition bit is off, the CPU directly executes the instruction. The second bit is *floating point condition bit*, which does the same thing for the floating point conditions. This feature of the instruction word makes it very suitable for *multiprocessor computations*. There is also a 4-bit op-code for indicating memory transfer modes and a few conventional instructions, e.g., no-op. The remaining bits are for indicating the integer and floating point wild and mass registers. Memory pointer registers are considered as a part of mass registers' address space. However, their I/O has some subtle differences with mass registers. Since we use 64-bit data bus, the remaining 32-bit word would be an immediate operand or a memory address.

In GSA, there are different ways that we can define the structure of an instruction word. What we have explained was just a simple example. In fact, logical scalability of the proposed architecture is quite simple. For ambitious powerful CPUs, we can extend the 32-bit instruction word to a 64-bit one. We call it *internal scalability.* From a different point of view, the proposed CPU is similar to a memory chip. Its instruction word, which contains the addresses of condition, wild, and mass registers, looks like the address bus of a memory chip and its 32-bit operand is similar to memory data. This property grants the ability of multiprocessing to the proposed CPU, in a new manner. It could be done in a similar way that is used to increase the capacity of a memory module by connecting several memory chips to a single bus. We call this property as *external scalability*. It seems that enough reasons have been presented to show why the proposed architecture is globe scale.

4. ROADMAP FOR CPU ARCHITECTS

Although the GSA removes many performance barriers from CPU architecture, memory latency is still a major obstacle. Since the only instruction type in GSA is data transfer, it exposes the ugly feature of this obstacle more than before. Current memory technologies enable us to achieve a high bandwidth, but not a low latency. In conventional architectures, we have two major methods to decrease the latency: caching and reordering. It is quite evident that the reordering has no meaning in GSA, so the only applicable approach would be caching.

Today's common approach in caching is to bring L1 and L2 caches on CPU die. It is good, but some better methods could be used. One approach would be the addition of a programmable cache memory to the CPU die. Another approach is to add a small fully associative cache memory to DRAM modules. Using this trick, a DRAM module is seen like an SRAM from the memory controller side. In fact, memory controller can read and write from/to the proposed cache memory. Afterward, the data are transferred between the proposed cache and DRAM chips using today's memory signaling system in a high transfer rate. Combining the two recent approaches is a good way for saving a GSA-based CPU from the memory latency disadvantages.

An important issue in GSA is the degree of compatibility with the former approaches like CISC, RISC, and EPIC. The RISC approach because of its reduced instruction set has the most compatibility with the GSA. It is possible to compile RISC assembly codes for a GSA-based CPU, easily. However, it is required more works for doing the same thing for CISC assembly codes, and please forget EPIC codes!

Finally, we should mention that the GSA is not only an advanced method for parallel processing or computational power boosting, but also it has an interesting impact on signal and image processing, AI, and sensor fusion applications.

5. COMMENTS FOR THE SECOND EDITION

In the new edition of this paper, we made no change to the content of the first edition. It is due to the passing of a very cool period (2000-2004) in the field of CPU architecture. Four years after the publication of the first edition, the poor presence of EPIC CPUs in the market can be seen apparently, which was predicted by the author. After the first publication of this paper, we received some criticism (some of them from a pioneer computer architect) about our presentation style, including the reference-less nature of this paper. In this edition, we eased some of the previous complicated sentences and also corrected the second figure.

REFERENCES

It should be mentioned that most papers of our Appetizer series are reference-less. As our Appetizer papers are more a critique of current industrial trends, we eliminate the references for not pointing to a few names in the industry as devil. We also suppose that the reader of these series is among the industrial activists.

ATG Press Appetizer

Volume 1, Number 2, December 2001

Important Note: The information, which is provided by this document, is devoted to the entire world for improving creativity, and therefore developing new job opportunities around the globe, in the field of computer science and engineering. So, no part of the proposed information is protected by any intellectual property right. However, the document itself, is copyrighted by Aria Technology Group. No part of this document could be published, in any form of media, without written permission of Aria Technology Group. This is a delayed publication. The actual release date is December 2004.

Globe Server

Farnad Laleh
Aria Technology Group

ABSTRACT

The popularity of a technology does not always help to its improvement. Among different technologies, application server is a good example. The Internet boom of the 1990s absorbed many attentions and resources to the development of more advanced application servers, which are the building blocks of the modern Internet. Since most servers on the Internet are information centric (versus computation centric), the result of those developments has been a big generation of standards for data and information representation. The standards by themselves, cannot improve the core technologies of application servers, either information or computation centric. In this paper, we present a software architecture for a typical application server which covers both information and computation centric paradigms. We call it *globe server* and the goal of this architecture is the improvement of application server core technology. It is a quite different concept from the standards for data and information representation.

1. INTRODUCTION

When a technology becomes popular and a growing demand backs it, introducing more advanced form of it in a minimum possible timeframe is inevitable and the natural trend of market driven technologies. If adequate basic research does not back that technology, it will be misdirected to a path that will be somehow dispraising if not an actual dead end. It is exactly what happened to application server technology in recent years. A historical outlook could help to understand this better.

The popularity of the Internet and its dominant data transmission protocol (HTTP) and information representation format (HTML), motivated the industry to develop new technologies, which were based on or derived from these two mainstreams. This derivation was not problematic by itself, but the problem came from confusing two strategies for these developments: *generality and universality*.

A concept (subject or object) is *general* if it is usable in different domains, in a same manner. For example, *byte* is a general concept in computer science and can be used in different computer science domains (artificial intelligence, computer graphics, databases, and many mores) in a same manner. But a concept is *universal* if it covers all aspects of a special domain while it would be useless in other domains. For example, Unicode is a universal format for text representation because it can represent text in all of languages (aspects) while it is useless in computer graphics or artificial intelligence.

Industrial improvement usually depends on the definition of new universal concepts not general ones. The reason is very clear: general concepts have different interpretations among different parties and it is hard and expensive to converge these interpretations into a single viewpoint, but universal concepts bring previous diverged concepts into a single viewpoint, which consequently make the future developments coherent and cost efficient.

HTML is a universal format for text and image representation. Its universality helped to its fast and growing popularity. This popularity motivated the industry to develop something that would be more universal than HTML. Due to the HTML popularity, there were many HTML-based tools around and for saving development cost and time, the industry speculated to reuse them in the new developing formats. So the new major formats, SGML and XML, were derived from HTML. Unfortunately, the lack of enough

basic research in the field cast these new formats as the general ones rather than the universal. However, XML is wrongfully introduced as a universal data format in some documents.

As a result, everyone can see divergence in the field. There are many vocabularies for XML, which try to standardize themselves as the future mainstreams. More interesting, we can see different viewpoint to the XML itself among academia and business sector. In business sector, XML is the base tool for service integration (leveraging XML-RPC, SOAP, and WSDL) while in academic world it is the base of the so-called Semantic Web (leveraging RDF and DAML+OIL) and autonomous machine-based activities on the net. In addition to these, the handling of large XML files has been remaining a great challenge.

In this paper, we establish a new universal software architecture for application servers, which can cover both of information and computation centric models. In section 2, we present the basic concept of the proposed architecture in an analogical explanation. In section 3, we present the exact technical model of the architecture. Finally, in section 4, we draw a roadmap for the future developments of globe server.

2. THE CONCEPT BEHIND OF GLOBE SERVER

In this section, we try to give a very simple analogical argument to introduce the concept of globe server. We suppose that the reader of this paper is familiar with the concepts like client-server architecture, two-tier, three-tier, and n-tier server architectures.

Today's application servers are much like a grocery. A client goes to a grocery for obtaining some stuffs. The grocer (presentation logic tier) is behind the counter and is in direct contact with the client and gets the client's order and gets back his/her required stuffs. The available stuffs (information materials) are categorized in the shelves (database) behind the grocer and are only accessible to him/her. If the grocery is small enough then the grocer can get the client's order and fetch the required stuffs from the shelves, lonely (a two-tier architecture). If the grocery is so large that one grocer cannot do the both jobs at a same time, then two grocers are needed (a three-tier architecture). The first is behind the counter and gets orders (gets back stuffs) from the client (presentation logic tier) and the second processes the orders and fetches the required stuffs from the shelves (business logic tier). If the grocery is much larger so that two grocers are still inadequate, then several grocers (an n-tier architecture) could share the jobs in their own manner. More than one are behind the counter and more than one process the orders and fetch the stuffs.

Now suppose that our client goes to a supermarket for obtaining the required stuffs. At a supermarket, our client can see all the shelves and stuffs and walk among them and select what he/she wants without ordering to another person. However, some salespersons may stand around the shelves to help the client in his/her decision and also control possible unauthorized access to the shelves. At the supermarket, the client has his/her logic (business and presentation logic) in walking among the shelves and accessing to the stuffs, while some supervisors may monitor this access. G-*server* (abbreviated form of globe server) is quite like a supermarket.

Technically speaking, when our client goes to a grocery and call the grocer, the client makes a *procedure call* with some parameters (his/her order) passes to a procedure. The grocer activity for processing the order, is the body of procedure and is not defined or affected by the client. In contrast, the client makes no procedure call at a supermarket; rather he/she does what it wants (at a controlled environment) to obtain the required stuffs. We call this *procedure definition*. So the key concept behind of g-server is procedure definition, which we explain it technically in the next section.

3. GLOBE SERVER ARCHITECTURE

Traditional application servers are the host of several procedures or services (as called these days), which are remotely callable by clients. The client job is to send some parameters to remote procedure(s) and gets back the result(s) instantly or after a period of time. These procedure calls vary from the simple HTTP Get

and Post to the more advanced SOAP and XML-RPC methods. In traditional application server architecture, it is the duty of server architects to predict the client needs and design the required procedures, accordingly. But as the volume of information on the server grows, it is very hard to predict the client needs and behavior if not impossible.

A recent approach is to design a group of different procedures (services) for different needs and then describe and introduce them to client using WSDL and UDDI. It is quite evident that this approach is expensive (because of the number of the procedures should be developed) and not efficient in highly dynamic or very large information spaces. Remembering the previous section, this approach is still like a grocery but a large one with a few grocers.

In g-server architecture, the server gives the ability of request description to its client. In this mechanism, which is called *remote procedure definition*, the client can describe for the server, the information it has, the information it needs, the format and style of the response, and a few other queries. It is performed employing a remote server-side scripting language that is called *globe server script language* or *g-script*, as abbreviated by us. G-server accepts the g-scripts that are generated (written) by client, so it can produce the exact response, at the right time, and in a customized form for its client.

We should now clarify two new concepts, the g-server architecture and the g-script language structure. The following figure shows the architectures of a traditional three-tier and a basic g-server. A few comparative notes would help to understand the g-server architecture.

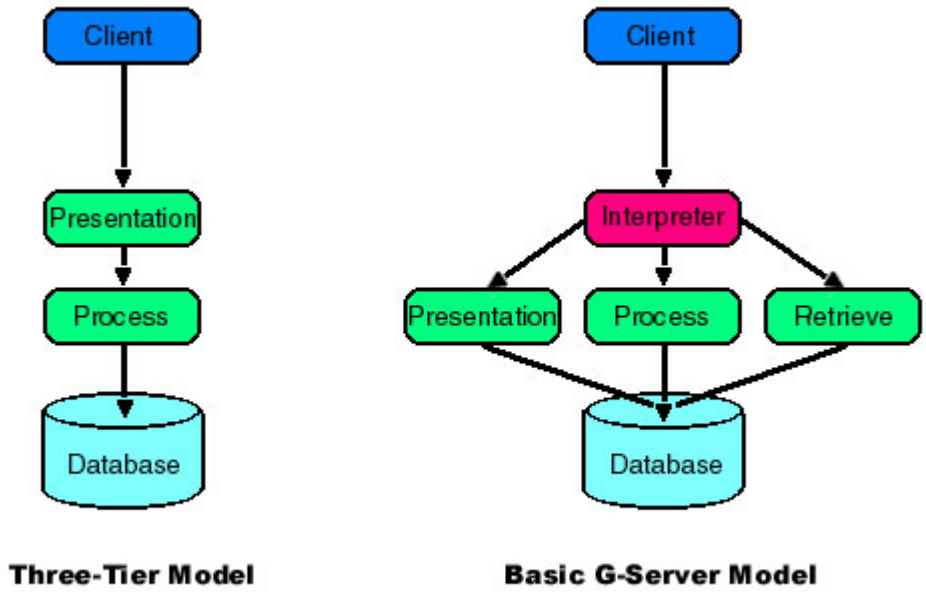

Fig. 1. Traditional three-tier architecture versus basic g-server architecture (the arrows would be bi-directional)

As can be seen, the traditional three-tier model has a serial architecture, i.e., the client is in touch with the presentation tier while the process logic (or business logic as stated these days) tier is an intermediate between the database and the presentation tier. The client can access the process logic only through a set of *design-time* embedded interfaces in the presentation tier. The process tier has the same role between the presentation tier and database.

In the basic g-server model, the presentation and process tier are not connected to each other serially. So in this architecture, we do not reckon them as tiers where they are *component packages*. Each component package consists of a library of different object-oriented components, which are recognizable for the interpreter and can be accessed by it.

The interpreter is in direct touch with the client and interprets the coming g-script from the client-side. Accordingly, the interpreter calls the required methods at the corresponding components to serve the client's request. There is a new component package in the basic g-server model that has not a corresponding tier in the traditional three-tier architecture. The retrieve component package is for searching among the existing information in the database and gives an extended ability to the client to retrieve its required information. This component package has a key role in large information centric servers while the role of the process component package is more important in computation centric servers.

In traditional application server architectures when a client makes a simple call (empty call) to the server, it gets back the required basic information from the presentation tier. But in g-server, client is not in direct touch with the presentation component package and should tell to the interpreter what it wants. This issue could be resolved by a very simple method, which is called *default scripting*. If a client sends an empty g-script to a g-server, the interpreter uses a few pre-defined and reserved g-scripts to serve the client.

The basic g-server as shown in figure 1, has three component packages. It should be mentioned that there are more complex models of the g-server that consists of more component packages. So we categorize the g-severs into three main models: basic, standard, and advanced.

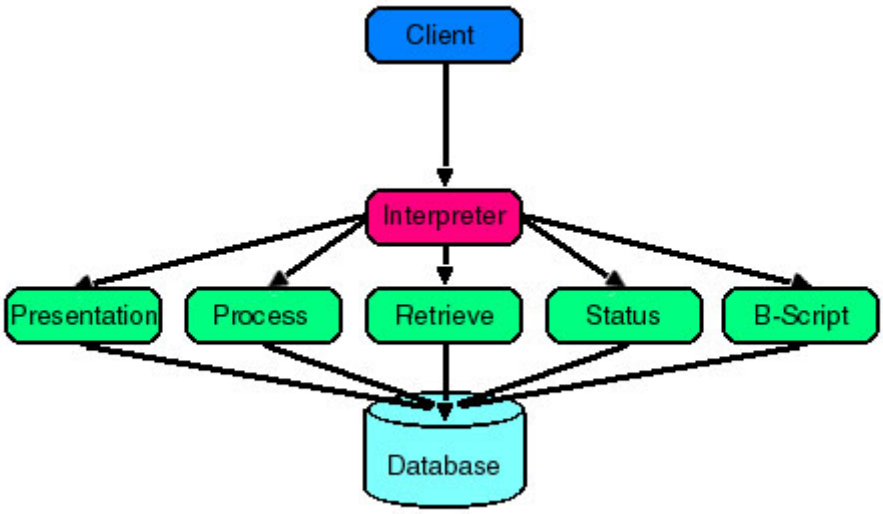

Fig. 2. Advanced g-server architecture (the arrows would be bi-directional)

Here is how the g-server is categorized in terms of its component packages. The minimum number of component packages for a g-server is three, which is its basic model.

1. The basic model includes the presentation, process, and retrieve component packages.
2. The standard model includes the basic g-server model plus the status component package.
3. The advanced model includes the standard g-server model plus the b-script component package.

There are two new component packages, see figure 2, which should be explained here. The status component package gives the ability of inter-relationship between the clients who use a same g-server. It includes some status flags (simple or complex string-based flags) that are set and read by the client, but under a predefined authorization policy. This inter-relationship has many applications in collaborative or competitive environments, e.g., financial markets, game nets, and many mores.

The b-script or *backward script* component package is a more complex concept. Backward script is generated in the server-side and implemented on the client-side. It should not be mistaken with the former familiar concepts like Java and VB scripts. B-script is the key concept behind of *globe client*, which is the subject of the next paper of our Appetizer series. So we do not explain it more in this paper except this useful note that b-script is the main road for autonomous and intelligent machine based activities on the net. In fact, b-script in conjunction with g-script can cover all of the goals of the Semantic Web.

4. ROADMAP FOR GLOBE SERVER DEVELOPERS

As g-script is a unique way of communication between different clients and g-severs, it should be standardized while the component packages and the interpreter can be developed on different platforms. The interfaces between the interpreter and component packages may or may not be standardized. It is a development strategy and is not a universal issue.

The only universal issue that should be addressed is the standardization of g-script. This is one of the two goals of ATG Caspian Hours Project. In this project, we try to establish a universal formats for g-script in collaboration with other institutions. In another part of the project, we do the same thing for b-script.

In addition to the powerful technical aspects of the g-server, which covered in the previous section, it is really interesting to mention that it has many useful applications in business. A party who holds a g-server can sell some g-scripts to the third-party clients for some special services. For example, a stock brokerage company can sell some special g-scripts to its best clients for giving them some special offers, whenever they want. In fact, a g-script can be customized for a client according to its character. This customization is a business itself!

Finally, it should be mentioned that the XML-based technologies, as the general technologies, may be used in g-server related developments. The only criteria for doing (or not doing) this, is the efficiency.

REFERENCES

It should be mentioned that most papers of our Appetizer series are reference-less. As our Appetizer papers are more a critique of current industrial trends, we eliminate the references for not pointing to a few names in the industry as devil. We also suppose that the reader of these series is among the industrial activists.

ATG Press Appetizer

Volume 1, Number 3, December 2002

Important Note: The information, which is provided by this document, is devoted to the entire world for improving creativity, and therefore developing new job opportunities around the globe, in the field of computer science and engineering. So, no part of the proposed information is protected by any intellectual property right. However, the document itself, is copyrighted by Aria Technology Group. No part of this document could be published, in any form of media, without written permission of Aria Technology Group. This is a delayed publication. The actual release date is December 2004.

Globe Client

Farnad Laleh
Aria Technology Group

ABSTRACT

Current application client technologies (both of thin and rich) are immature when compared to application server technologies. The immaturity is originated in different technical and commercial issues. Technically, application clients have not strong and universal definitions and standards (e.g. something like HTML) as application servers have. Commercially, business sector represents itself through its application servers, so the development efforts are diverted to the server-side projects. From the engineering point of view, since the overall performance on a network depends on both of client and server, the client-side developments are as important as the server sides. In this paper, we try to categorize technically plausible application clients into a few comprehensive classes and put a definition upon each class. Afterward, we show a way in which each class can be standardized for the future coherent developments. Finally, we highlight the classes that have a remarkable potential for the current and the future commercial applications.

1. INTRODUCTION

The concept of application client is tied with the concept of distributed computing systems. In traditional distributed computing, application client is an application program that uses the resources of both the underlying computer and other accessible computers on a network. In recent years, this concept is referred to as the *rich application client.*

The advent of the Internet and its explosive growth pushed the usage of a new group of clients. These clients were targeted to show the information which was cast in HTML format, i.e., the dominant information representation format on the Internet. Since the HTML-based representations could not exploit many features of the ready-to-use application program interfaces (APIs) on the client-side, some new technologies (like the Java virtual machine) were added to the proposed clients. This could help more usage of the client-side resources but is still far from the optimal usage of the proposed resources. So this group of client is referred to as the *browser-based application client* or the *thin application client.*

This traditional categorization of application clients is apparently based on the usage degree of the resources on the client underlying system. This sort of categorization can only address the designing of *development tools* for application clients not the development itself. In fact, it does not draw a roadmap for the development of different possible application clients and leaves it as a quite general issue to the developer.

We think that application clients could be categorized based on their own functions. This categorization not only opens some new ways for the universal development of application clients, but also establishes a strong base for the development of autonomous and intelligent application clients. The recent issue (intelligent clients) has been followed in different forms in the past 10 years. We can point to agent-based systems and the Semantic Web as the two major efforts toward this, but both of them have been remaining as the general concepts (see Globe Server in Appetizer series) rather than the universal ones.

In this paper, we categorize application clients based on their functions, in section 2. Afterward in section 3, we present a universal software architecture for application clients. This universal model, which is called *globe client*, covers all of the categorized classes. Finally, we draw a roadmap for the future developments of the globe client and highlight its commercial applications in section 4.

2. CATEGORIZATION OF APPLICATION CLIENTS

There are different types of application client around for years, ranging from the older FTP and SMTP clients to the recent video conferencing and messenger systems. Although these clients seem substantially different from the application point of view, but there are based on a few basic functions from the engineering point of view. We use these functions as criteria for our categorization as the following:

1. **Information representation:** client represents to its user the information which are presented by server. For example, a Web browser represents images and text, which are cast by the server in an HTML file.
2. **File transfer:** client sends and receives file to/from server. In addition, we put video and audio streaming into this category. FTP clients are a simple example for this category.
3. **Message transfer:** client sends and receives message to/from server. The messages could be text, voice, and video. Chat clients are a good example for this category.
4. **Information process:** client processes the information it receives from server and produces a result, accordingly. The result may be sent back to server or used by the client itself. This is the most general category and entails many activities from simple form processing to the more complex scientific computing and artificial intelligence procedures.
5. **Network traversal:** client may traverse the network space for finding its required resources on a real or a virtual server. A virtual server is a client that can share some resources with a limited number of other clients. This category is not widely in use these days, but it is an important part in the future applications, especially in aggregated and intelligent environments.

It should be mentioned that the user interface of a client is not considered as a function of it, i.e. we can change the client user interface while its function does not change. In this paper, what we call as a *client function* is one of the above five categories. A client can have more than one function or may have all of the five functions at once. In addition, these functions may have inter-relationship inside a client. *Globe client* is an application client that contains at least three of the above functions. In the next section, we explain its architecture and its different models.

3. GLOBE CLIENT ARCHITECTURE

Currently, we are familiar with the possible functions of a typical application client, so we can propose a universal software architecture for it. As it was mentioned before, we do not consider user interface as a part of the client functions, although it is a part of client architecture. In addition to the well-known concept of user interface, g-client (abbreviated form of globe client) uses a new sort of interface, which we call it *server interface*. This new interface is a powerful technology that allows a server to talk to its client in order to help it for performing its functions.
A simple example would clear the concept of server interface. Suppose that a client represents to its user an identification form it receives from a server. The user should fill and send it to the server but forgets to fill it completely and sends it incompletely. In traditional client-server applications, the server returns an error message to the client, which is represented consequently to the user. Employing the server interface technology, the server tells to the client the incomplete part of the form and asks if the client itself can complete it for the server. If the client knows the incomplete part and has the authorization for sending it to the third party, then it completes it for the server without the user interference. If not, it shows an error message to the user.

The server interface technology is the base of automatic client activities. We should mention that globe client is substantially different from *intelligent client*. An intelligent client is based on some inference

engines and may do some functions in its favor. Contrary, a g-client does exactly the functions that its server asks it and these functions are toward the user requests and policies. G-client does not use inference and other AI-based mechanism in its server interface technology while it can use these methods as a part of its information processing (the 4th category) function.

The key concept behind of server interface is the *remote procedure definition*, which we explained it before in Globe Server (see Appetizer series). We use this concept same as the way we use it in g-server. The only difference is the substitution of client and server, i.e., g-client accepts remote scripts that are generated by server so it can produce the exact response, at the right time, and in a customized form for its server. Since this sort of script comes from the server-side after the client request to server, we call it *backward script* or *b-script*. The following picture shows the architecture of g-client.

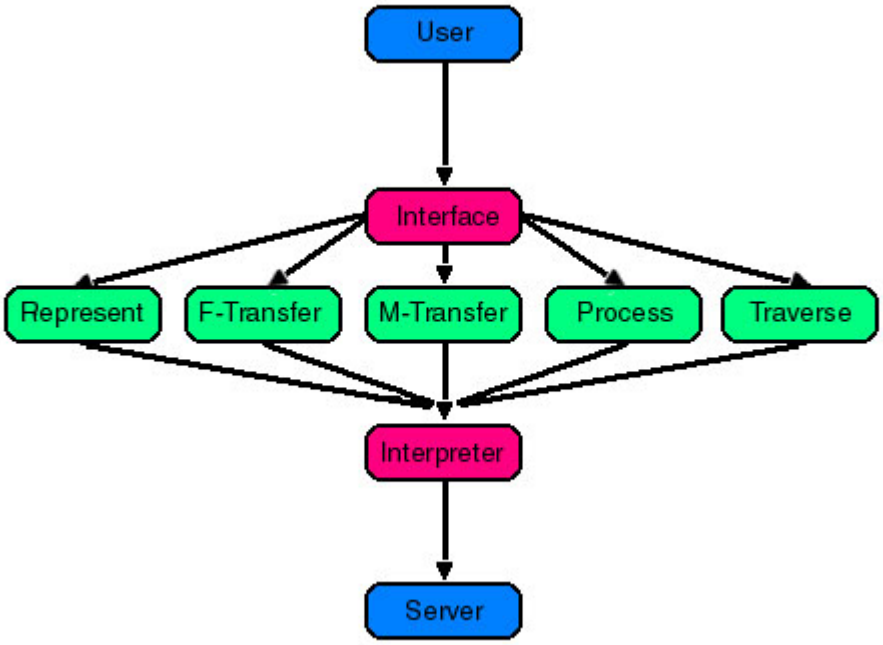

Fig. 1. Advanced g-client architecture (the arrows would be bi-directional)

The picture shows that the user is in direct touch with the interface (user interface) as the server is with the interpreter. The interpreter has the role of server interface and interprets the coming b-script from the server-side. There are five *component packages*, which correspond to the five mentioned functions in the previous section. Each component package consists of a library of different object-oriented components, which are recognizable and accessible by both of the interpreter and the interface. The represent, f-transfer, m-transfer, process, and traverse component packages correspond to the information representation, file transfer, message transfer, information process, and network traversal client functions, respectively. We categorize the g-client into three main models (as we did for the g-server before) as the following:

1. The basic model includes the represent, f-transfer, and m-transfer component packages.
2. The standard model includes the basic g-client model plus the process component package.
3. The advanced model includes the standard g-client model plus the traverse component package.

If you remember the advanced g-server architecture, it has a component package named b-script, which is supposed to generate the required b-scripts for g-client. Contrary to this, we have no corresponding components package in the g-client for generating g-script. It does not mean that g-client cannot generate any g-script, rather it does it using the process component package and the generated g-script is based on the result(s) of information processing function.

Another important issue which should be addressed is the position of former technologies like the Java Virtual Machine (JVM) in g-client architecture. Fortunately, their positions are very clear. These technologies are embedded in their corresponding component packages, e.g., the JVM is embedded in the represent component package. Again, a concept like the Java script should not be confused with the concept of b-script. They are in two different separate layers.

4. ROADMAP FOR GLOBE CLIENT DEVELOPERS

As b-script is a unique way of communication between different servers and g-clients, it should be standardized while the component packages and the interpreter can be developed on different platforms. The interfaces between the interpreter and component packages may or may not be standardized. It is a development strategy and is not a universal issue.

The only universal issue that should be addressed is the standardization of b-script. This is one of the two goals of ATG Caspian Hours Project. In this project, we try to establish a universal formats for b-script in collaboration with other institutions. In another part of the project, we do the same thing for g-script.

In addition to the powerful technical aspects of the g-client, which covered in the previous section, it is really interesting to mention that it has many useful applications in business. For example, a consulting firm can prepare different b-scripts for its different clients. The b-scripts assist the clients in interaction with the firms. In addition, the firm can give some information about other firms to its clients through the b-scripts.

Suppose that a client decides to buy 10,000 barrels of 40-degree oil from its long time partner, broker A, so the client (using its g-client) sends a request (g-script) to the broker g-server. Since the broker has not more than 6,000 barrels available for now, it sends the address of broker B and C g-servers along with an estimated price through a b-script to the g-client to find the remaining 4,000 barrels at the most competitive price. Afterward, the g-client sends two separate g-scripts to the broker B and C g-servers according to the received b-script from broker A. If one or both of the B and C offer a price bellow the estimated price of broker A, the g-client reports the situation to its user to decide for a buy.

Scenarios like the above are the major goal of the Semantic Web and the agent-based system researchers. While these fields highly depend on artificial intelligence and other bottlenecks in computer science, we can reach the same goal with the old-fashion and easy to use scripting systems. People sometimes come to trouble, because of not facing a hard problem, but not knowing what is the problem!

REFERENCES

It should be mentioned that most papers of our Appetizer series are reference-less. As our Appetizer papers are more a critique of current industrial trends, we eliminate the references for not pointing to a few names in the industry as devil. We also suppose that the reader of these series is among the industrial activists.

ATG Press Appetizer

Volume 1, Number 4, December 2003

Important Note: The information, which is provided by this document, is devoted to the entire world for improving creativity, and therefore developing new job opportunities around the globe, in the field of computer science and engineering. So, no part of the proposed information is protected by any intellectual property right. However, the document itself, is copyrighted by Aria Technology Group. No part of this document could be published, in any form of media, without written permission of Aria Technology Group. This is a delayed publication. The actual release date is January 2005.

Globe Optimizer

Farnad Laleh
Aria Technology Group

ABSTRACT

The field of combinatorial optimization and integer programming has a rich literature but a poor progress. It is due to the limited number of fundamental concepts which are applicable in the development of the field, so the major research efforts go through the different combinations of these concepts to reach for a slightly better result. Since it has been proved that most problems in this field are in NP-C class, almost all of the efforts are diverted to finding good approximating methods to the solution of these problems. A few heuristic methods from the older greedy approaches to the newer genetic algorithms are the engines of these approximations. There are also a few exact methods like the dynamic programming and the branch and bound, which have a quite narrow applicability. In this paper, we introduce a new original method to the solution of combinatorial optimization problems, which is called *coordinated brute force* method. Basically, It is an exact method to the solution of NP-C problems, but it can be converted into a record breaking approximating method by a few slight changes. We apply this method to three famous NP-C problems: the traveling salesman as the hardest NP-C, the facility location as a strong NP-C, and the knapsack as a non-strong NP-C.

1. INTRODUCTION

Computer science is an interesting scientific field because of its unique nature: it highly depends on the mathematics and logics so that many scientists believe it as a branch of the mathematics while its progress engine is the intuition (like the arts) not the math. The algorithm as the core concept of computer science is based on the intuition as its *constructor* while the math is used as its *analyzer*.

Among different computer science fields, the filed of combinatorial optimization has a distinctive specification: there are a few basic algorithms to the solution of the problems in the filed while there are many sophisticated analyses around the problems. These analyses include a range of different stuffs from the classification of the problems to the *reduction theory* and *approximability*. Although these stuffs open a wider perspective to the problems but they could not originate new solutions to them.

Addressing the above issue, we introduced *coordinated brute force* method in 1997 [1] but not under this label. Due to the alternative nature of the proposed method, we only introduced its concept intuitively and without any name to avoid any possible misleading. However, the lack of labeling made it hard to categorize the method and the feedbacks we received from researchers during the past eight years prove this claim. Therefore, we decided to explain the concept of the coordinated brute force method in a more complete form. We also give some comparative studies between the proposed method and other classic methods (like the greedy approach) to make it easy for researchers to categorize it.

Since the tradition of this field directs us to analyze the proposed method, we open a way to do this. The unique feature of this method is its hybrid nature: this method is an exact solution to the NP-hard problems (so it is not in polynomial time) while by eliminating a single process in the method, it appears as an perfect approximating method in polynomial time. As this method is a universal one (can be applied to the most of NP-hard problems), a general *approximation threshold* [2] cannot be considered for it and it depends on the case of each problem. In this paper, when we use the term “approximating method” it does not imply the existence of an *approximation scheme* [2].

In section 2 of this paper, we introduce the concept of the coordinated brute force method by introducing it as an intuitive solution to the traveling salesman problem. In section 3, we apply the proposed method to the facility location and the knapsack problems and analyze the performance of the proposed solutions. Finally, in section 4, we discuss how it can be applied to other NP-hard problems and how its performance can be increased when we consider it as an approximating method.

2. COORDINATED BRUTE FORCE METHOD

The brute force algorithm has a very simple concept: try all possible ways and check the results. However, it comes to failure in the case of NP-hard problems as the number of possible ways grows exponentially. The interesting point in the brute force algorithm is that the technique we should use to select all possible ways is an open choice. This is exactly what the coordinated brute force method wants to address. In this method, we select the best choice at first, the first runner up at second and so on. It sounds like a joke: if we know what is the best choice, why should we solve the problem? The answer is we do not know what is the best choice, but we can know it sooner than every other possible choice! The following intuitive approach to the TSP clarifies this concept.

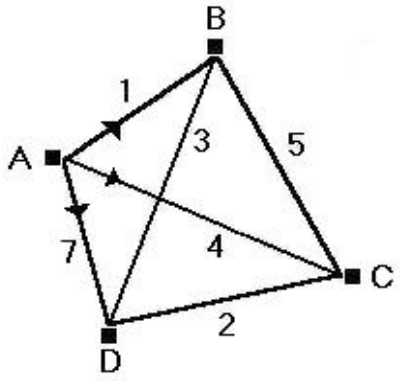

Fig.1. A 4-city non-Euclidean TSP sample

Suppose we are required to find the shortest tour starting from city A in figure 1. Our approach is to send a laser beam from city A to all other cities (e.g. through a fiber optic) simultaneously. As the sent beam from city A is received by any other city, it is modulated in order to indicate the traversed path by the beam. We suppose that the modulation time is negligible and equal to zero. The modulated beam is transmitted again to the other cities. The first beam that has gone through all the cities and reaches the home, i.e. city A, indicates the shortest tour.

As all possible tours are tested in the above approach, it is clearly a brute force approach but a *coordinated* one, i.e., we receive all possible choices in an ascending form. It sounds like a magical solution but it is still an approach and not a complete solution, i.e., we have not proposed any method for the implementation of this approach on a conventional computer (Turing machine). Fortunately, the projected implementation is as simple as the concept itself, using a new powerful concept which is called *space tape* [1]. In fact, space tape is a sequenced view to the proposed approach: if you cannot work in parallel to send beams simultaneously, you can send them sequentially but in their *corresponding* dispatch time and your memory can be used to register the proposed dispatch times.

To implement this concept, we consider a two-dimensional (width × length) memory array as a space tape along with a pointer, which can point at any given time to a position in the length of the tape. This pointer is somehow similar to the pointer in Turing machine but it can move only to forward. The space tape is used to register (store) the beam traveled path and the beam arrival time in each city (dispatching time from that city) in its width and length, respectively. Now we can implement the proposed approach on a conventional computer for the sample in figure 1, as the followings.

Consider city A as the starting city of the shortest tour and the pointer position as zero. The distances from city A to each of the other cities are determined and the corresponding beam path, which we call it sub-path, is registered in the horizontal position that corresponds to the traveled distance of the sub-path and the vertical position corresponds to its arriving city. For example, AB is registered in length 1 and width B, and AC in length 4 and width C. This *allocation* is shown in figure 2. After the initial allocation, the pointer moves one step to forward and comes to position 1. We check the width of the tape in length 1 to see if any sub-path is registered or not. For any registered sub-path, which is only AB in this example, the sub-paths which are started from city B and are not dispatched to the previously traveled cities, are registered in their corresponding position as explained before. Afterward, the pointer moves to the next position. Figure 2 shows the tape situation when the pointer arrives in position 2.

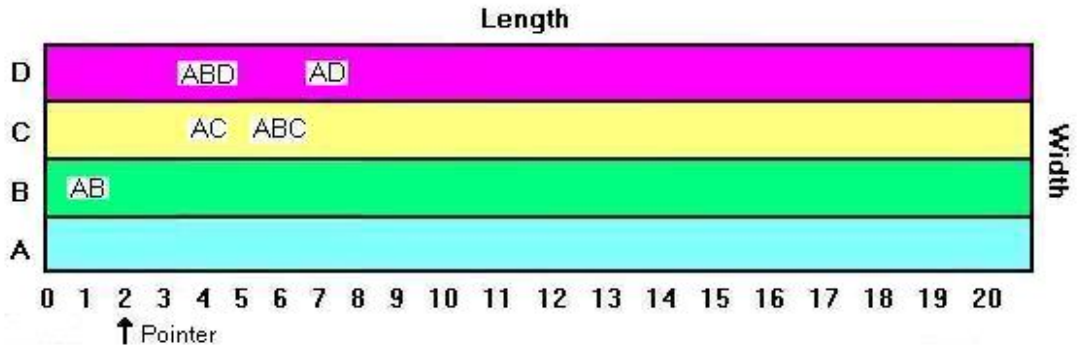

Fig. 2. Space tape situation for the sample in fig. 1 when the pointer arrives in position 2

The mentioned allocation is repeated until the pointer arrives in a position in which a complete tour can be detected. This tour is our required answer and the length in which it has been registered, represents the length of the optimal tour. For the proposed example, it happens in length 10 as is shown in figure 3. As can be seen, the optimal tour is ACDBA or ABDCA in its reverse form.

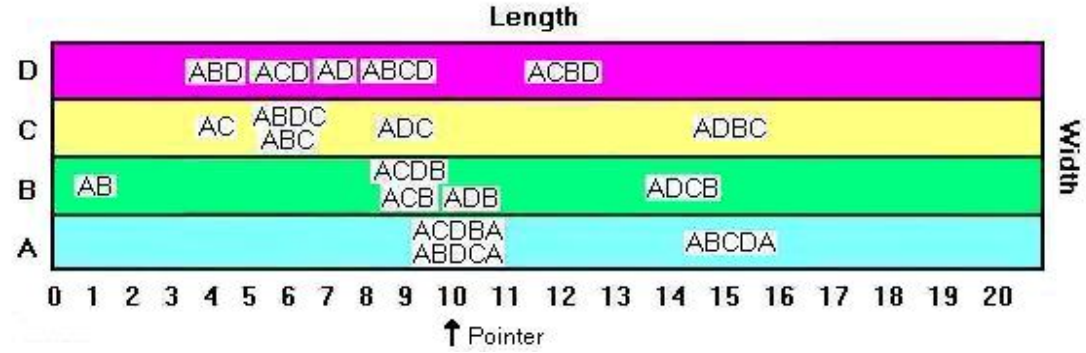

Fig. 3. Space tape situation for the sample in fig. 1 when the pointer arrives in position 10

The proposed algorithm solves the TSP in time $O(nWL)$, where n is the number of cities, W and L are the values of the width and the length of the tape, respectively. It may look like a *pseudo-polynomial* algorithm, but it is not because of the maximum required value of W. To keep this algorithm as an exact problem solver, the width of the tape should have enough space to register all of arriving sub-paths in any position of the length. As the volume of arriving sub-paths in any position does not exceed 2^n (in any position, you can register only one of the sub-paths whose starting, arriving and traveling cities are the same and disregard the others), the proposed algorithm is in time $O(n2^n L)$ which is not apparently a pseudo-polynomial algorithm. If it was, then P = NP [2].

The interesting point is that the dynamic programming can offer a solution to the TSP in time $O(n^2 2^n)$ [2] which is very similar to the time of the space tape method. In addition, if we put a limitation on the allocation in space tape, we reach the well-known Dijkstra's greedy algorithm [3] for the shortest path problem: once a sub-path arrives in a city, do not dispatch other sub-paths to that city. The result is all shortest paths from the starting city to all other cities. In this case, the algorithm is in time $O(n^2 L)$.

You may be surprised by the flexible nature of the space tape method: it looks like the dynamic programming, the greedy approach, and as we explain later, a perfect non-heuristic approximating algorithm! The secret behind this is the strong intuition in the coordinated brute force method. In the next section, we explain how more features of this interesting method could be exploited.

3. ESTABLISHMENT OF GLOBE OPTIMIZER

The space tape method can exactly solve different NP-hard optimization problems. However, the width of space tape grows exponentially in NP-hard problems, so we should find a way to reach for a feasible solution if we cannot reach the exact one. Fortunately, it can be achieved easily by limiting the growth of the width.

In the TSP, the width maximum value, W, can be limited by order of n^2 (sub-paths are categorized by their arriving city and in each arriving city by their number of traveled cities) to obtain an $O(n^3 L)$ approximating algorithm. In this case, if two sub-paths compete for one place in space tape, we should arbitrary eliminate one of the two. If two sub-paths have the same traveled-city histories, the elimination has no effect on the exact solution and so it is called *white elimination*, if not, it is called *gray elimination* because of its potential to disrupt the exact solution. For a given sample of the problem, if we apply the approximating algorithm and a solution without any gray elimination is generated, then we can be sure that it is the exact solution! It is another interesting feature of the space tape method.

Since the TSP is NP-hard, all other NP problems can be *reduced* to it and get a feasible or exact solution by the space tape method. This reduction has a polynomial time overhead and it is more interesting to solve other problems directly by the space tape. Fortunately, the flexible nature of this method makes it possible to apply it directly to the most of combinatorial optimization problems. This is why we call it *globe optimizer*.

The second target problem is the *uncapacitated facility location problem* (UFLP). In this problem, we are given a set of n cities (clients) and a set of m facilities (service points). Every facility has a *setup cost* (opening cost) and between every city and every facility, a *connection cost* is considered. All costs are nonnegative integer values. We are requested to find a subset of the facilities for which the total cost (setup costs plus the costs of connecting every city to an open facility) is minimized. It is apparent that each city should be connected to its nearest open facility, so the problem is to find the best subset of the facilities.

At the first look, the problem seems to have nothing to deal with the space tape. The key point to the solution of any problem using the space tape is the ability to define the structure of sub-paths for that problem. In the UFLP, a sub-path is a string of the names of opened facilities and each position in the string corresponds to a city. Therefore, the maximum length of a sub-path is n (the number of cities). For example, if we have 10 cities marked 1 to 10 and 6 facilities marked A to F, then a complete sub-path would be BAACFACFBC. The proposed sub-path denotes that cities 2, 3, and 6 are connected to facility A; cities 1 and 9 to B; cities 4, 7 and 10 to C; cities 5 and 8 to F; also facilities D and E are not opened.

In the TSP, we cannot meet each city more than once and therefore the name of a city is not repeated in a sub-path (except the starting city). Contrary, we can do this in the UFLP because every facility can serve all cities. This is why the problem is named *uncapacitated* facility location. Each sub-path is registered in the length that corresponds to its total cost. The first complete sub-path in the tape is the solution.

The proposed algorithm is in time $O(mWL)$, where m is the number of facilities, W and L are the values of the width and the length of the tape, respectively. If we limit W by order of n (register every sub-path in a width corresponds to the last connected city), an approximating algorithm in time $O(mnL)$ is obtained. The fact that every city should be connected to its nearest open facility can be applied to this algorithm as a white elimination rule. In this case, a more powerful approximating algorithm in time $O(mn^2L)$ is achieved.

The third target problem is the *knapsack*. We select this problem as the last example for the space tape method because of two interesting features:

1- It is not a *strong* NP-hard problem [2] and therefore a pseudo-polynomial algorithm exists for it. We want to see how the space tape deals with this problem and what is the time order of the resulting algorithm.
2- In contrast to the two previous examples, it is a maximization problem.

In the knapsack, we are given a set of n objects and a knapsack of capacity C. Every object has a *size* and a *profit*. All of the values are positive integers. We are requested to find a subset of the objects so that they can be fit in the knapsack (their total size should not exceed C) and their total profit is maximized.

The structure of a sub-path for this problem is very simple. It is a string of the names of objects which are located inside the knapsack. For example, if there are 6 objects marked A to F, then ADF denotes objects A, D, and F are located inside the knapsack. The permutation of the objects does not matter, e.g., ADF and DFA are equal and can be replaced with each other in the tape. The length in which a sub-path is registered corresponds to the total profit of the sub-path. In addition, a sub-path is dispatched if its total size does not exceed C by adding the new object. Since this is a maximization problem, the solution is the last sub-path in the tape that cannot be dispatched anymore.

This algorithm is in time $O(nWL)$, where n is the number of objects, W and L are the values of the width and the length of the tape, respectively. If we limit W to 1, the resulting algorithm is not only an exact solution to the problem, but also a pseudo-polynomial algorithm in time $O(nL)$. This interesting limitation of W is done by applying a perfect white elimination rule: if two sub-paths compete for one place in the tape, select the one with smaller size. In this paper, we do not prove why this is not a gray elimination rule, but it is easy for the reader to do it.

The interesting point is that the dynamic programming offers a solution to the knapsack in the same time order [2]. More interesting, the dynamic programming cannot be applied to the solution of the more complex *multidimensional knapsack problem* (due to its exponential time order) [4], but the space tape method can do this (although the resulting algorithm is not a pseudo-polynomial one) as it did for the other strong NP-hard problems.

4. ROADMAP FOR GLOBE OPTIMIZER RESEARCHERS

The coordinated brute force method, or its Turing compatible version, the space tape, can be directly applied to many hard combinatorial optimization problems in a similar way to what presented in the previous sections. Among different problems, we can point to the Hamiltonian path, the bin packing, the job scheduling, the graph coloring, and the subset sum problems. We develop the globe optimizer for some of the mentioned problems in ATG Persian Gulf Project. Since we believe in $P \neq NP$ conjecture, we do not recommend researchers to use the space tape for bringing NP into P. Instead, we encourage researchers to develop new approximation schemes for the NP-hard problems using this method.

In contrast to the knapsack problem that has a *fully polynomial* approximation scheme, the TSP cannot have any approximation scheme unless P = NP [2]. However, the UFLP and many other NP-hard problems can have good approximation schemes. Please note that when we use the space tape as an approximating approach, it does not entail that we can determine an approximation threshold for it.

Having the coordinated brute force method around, the use of heuristic methods (like the *genetic algorithms*, the *neural networks*, the *phase transition*, and many others) to the solution of combinatorial optimization problems is somehow unjustifiable. The accuracy of the space tape method can be easily improved by increasing the tape width in order to reduce the gray eliminations. In addition, new white elimination rules may be found in some cases and this is the most wanted research activity in this field.

Finally, we recommend two simple methods to reduce the tape length (a common complexity factor for all samples) in memory. Firstly, the tape can be a circular memory array whose length is the maximum distance between two cities in the problem dataset plus one. Secondly, the numbers in a dataset can be normalized to reduce the length of the tape without any effect on the solution. For example, we can reduce the distance between all of the cities in the TSP by subtracting the minimum distance between two cities in a dataset minus one from all of the distances in the dataset. Anyway, if you apply the space tape method to your datasets, you will find that the tape length is not a challenge.

REFERENCES

[1] M. Kamali, F. Laleh, and A.R. Mirzai, "A New Parallel Approach to the Solution of the Traveling Salesman Problem," in *Proc. IASTED & AAAI Conf. on AI & Soft Computing*, Banff, 1997, pp. 308-310.

[2] C. H. Papadimitriou, "*Computational Complexity*," First Ed. Addison Wesley, 1994.

[3] N. Deo, "*Graph Theory with Applications to Engineering and Computer Science*," First Ed. Prentice-Hall, 1974.

[4] D. Pisinger, "An Exact Algorithm for Large Multiple Knapsack Problems," *European Journal of Operational Research*, 1999, pp. 528-541.