On the Notion of Parallelism in Artificial and Computational Intelligence

Benedek Nagy

Department of Computer Science, Faculty of Informatics, University of Debrecen
Egyetem tér 1, H-4032 Debrecen, Hungary
Research Group on Mathematical Linguistics, Rovira i Virgili University
Tarragona, Spain
nbenedek@inf.unideb.hu

Abstract: In this paper the appearance and the role of the parallelism is analysed in some algorithms and approaches of problem solving in Artificial Intelligence and in Computational Intelligence. Well-known that parallel algorithms can solve the problems in a more effective way than the sequential algorithms. Two basic types of parallelism are presented. The so-called 'or'-parallelism uses the parallelism to do brute-force search. In the so-called 'and'-parallelism the solution is built up by several parts given by the parallel branches of the computation.

Keywords: Parallel computing, Backtracking, Graph-search, Problem-reduction, Genetic algorithms, Simulated annealing, And-parallelism, Or-parallelism, Artificial Intelligence, GRID

1 Introduction

There are several hard problems. In complexity theory there are well-known NP-hard, P-SPACE and more complex problems. These complex problems cannot be solved in easy way; there is not any simple (say polynomial) algorithm which can solve any of them. One of the main tasks of Artificial Intelligence and Computational Intelligence to provide efficient way of problem solving. There are some traditional artificial intelligence (AI) algorithms, such as Backtracking search, Depth-first and Breadth-first graph-search algorithms and the Problem-reduction method. These algorithms have been implemented on traditional computers for decades. They usually work sequentially. There are some newer branches of AI called Computational Intelligence (CI). To make more efficient problem solving most of these new approaches involve some kinds of parallelism. Well-known CI approaches are the Simulated Annealing and the Evolutionary algorithms such as Genetic algorithms.

In this paper we are trying to characterize the notion of parallelism can be seen in these various problem solving approaches. In the next section the conventional AI algorithms are analysed. In Section 3 we deal with some of the CI algorithms. In Section 4 the basic types of parallelism are presented and the possible parallelism is shown in the detailed AI and CI algorithms.

2 Traditional (Conventional) AI Algorithms

In this section we recall the well-known AI problem-solving and search strategies, such as Backtracking, some of the graph-search algorithms and the technique Problem-reduction [4, 7, 8, 10].

2.1 Backtracking

Backtracking is one of the oldest computational strategies for finding solutions. There is an initial state. The algorithm steps to a successor state to examine it. If there is no such a state the actual state is deleted from the path and other non-examined successor will be tried.

This search algorithm use the tree form of the state-space graph, i.e. the states can be achieved in more than 1 way will be analysed by multiple times as it can be seen on Figure 1.

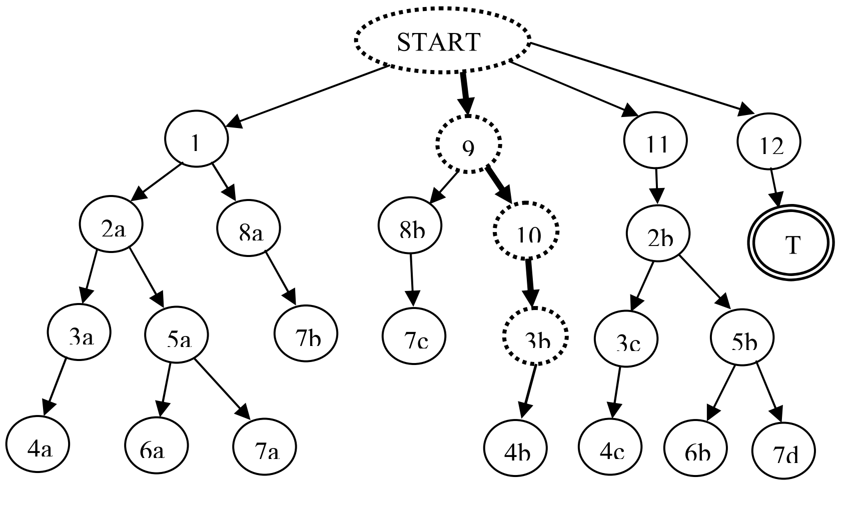

Figure 1
The tree form of the search-space used by Backtracking search

In the given example the nodes having the same number represent the same state. The state-space traversal can go in the following way: START, 1, 2a, 3a, 4a, backtrack, backtrack, 5a, 6a, backtrack, 7a, backtrack, backtrack, backtrack, 8a, 7b, backtrack, backtrack, backtrack, 9, 8b, 7c, backtrack, backtrack, 10, 3b (it is the actual in the figure), 4b, backtrack, backtrack, backtrack, backtrack, 11, 2b, 3c, 4c, backtrack, backtrack, 5b, 6b, backtrack, 7d, backtrack, backtrack, backtrack, backtrack, 12, T (final state, terminal node).

In Backtracking search algorithms there is only 1 path in the database at each time. Therefore traditionally it is a sequential approach and there is not any parallelism. To avoid the visit of the same node twice or more times and so, to save time, the graph-search algorithms give the solution.

2.2 Graph-Search Algorithms

A graph search algorithm begins at the start node and explores all the neighbouring nodes and put them to the list of open nodes while the explored node is removed from this list and appended to the list of closed nodes. Then for each of those nodes which are in the list of open nodes, the algorithm explores their unexplored neighbour nodes, and so on, until it finds the solution.

In these algorithms the examined states will not be deleted from the database, therefore the algorithm will not visit the same state more than once.

Although these algorithms work in traditional computers in sequential manner a form of parallelism is present in the following sense. The database consists of several paths from the start-node. In each step the operation expand means that the algorithm tries to continue the chosen path in all possible way, i.e. all non-visited neighbours of the end-state of the chosen path will be explored for the database.

The state-space graph of the previous example can be seen on Figure 2.

Depending of the order of the expansion of the nodes there are various types of the graph-search algorithms. If a node with the smallest deepness value is chosen to expand the algorithm works as a Breadth-first search algorithm, while if a node with the largest deepness value is expanded in each step the Depth-first search approach is used.

The state-space represented on the figure is examined on the following order by the Breadth-first algorithm: START, 11, 1, 9, 12, 2, 8, 10, T and it is terminated with the solution. Using Depth-first search the order of the explored states is somehow between the Backtracking and Breadth-first approaches: START, 11, 1, 9, 12, 2, 3, 5, 4, 6, 7, 8, 10, T.

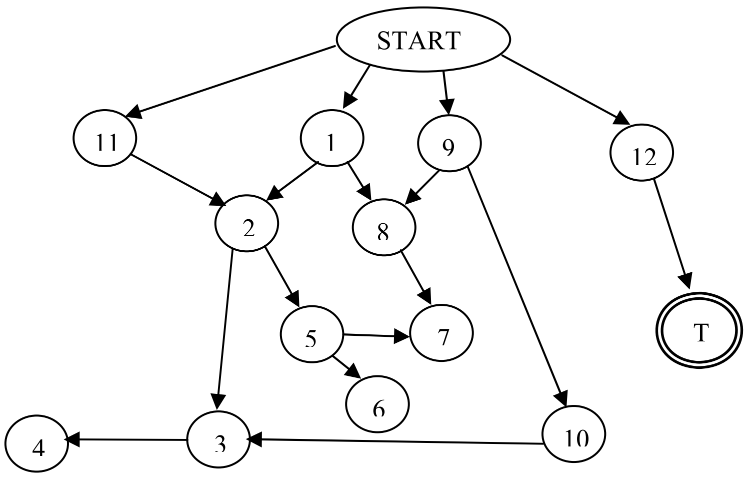

Figure 2
The search-space used by graph-search algorithms

2.3 Problem-Reduction

This problem solving method is also well-known. The original problem is in the root of an AND-OR graph. (Usually AND-OR graph is used in these reductions.) All other nodes of the graph contain some related problems, most of them should be easier than the original one. The leaf nodes are representing trivial or clearly unsolvable problems. The solution is given if there is a finite hyper-path starting from the root with the following property. Each node in the hyper-path either trivial or has an and-bundle of branches having all nodes with the same property.

The method uses a reduction of the original problem to some subproblems, and reduction of these subproblems and so on until only trivial problems are obtained. The solution of the trivial problems is trivial (coming from somewhere), and their solutions will be used to obtain the solution of the more complex and so, of the original problems. For instance, the solution of the problem of towers of Hanoi by this recursive method is well-known [4]. The game is to put all the n discs from the column A to C. It can be divided to the following subproblems: put n-1 smallest discs from A to B, put the n-th disk form A to C and put the n-1 discs from B to C. The first and the third tasks can be divided to smaller problems if $n>2$. The second task is a trivial one. The solutions of these subproblems together give the solution of the original problem. In this particular problem the representation goes by an AND graph.

The above described method is related to the known rule/method ‘divide and conquer’.

3 CI Algorithms

One way is to deal with intractable (say NP-hard or more complex) problems to put some kind of parallelism to the algorithm. Although these algorithms usually work on traditional computers too, they directly use some parallelism. They do not search all the state-space but only its some preferable parts. These preferable parts are chosen in special way depending on the approach. These algorithms can be used in very large problems (large state-space) where traditional AI algorithms cannot provide fast and good solutions.

There is no free lunch [12, 11], so if one wants faster result then he/she must dispense with the exact result and satisfied by a 'relatively' good solution. Most of the CI algorithms provide relatively fast solutions which may be not the best one(s).

In the following subsections we recall two kinds of CI approaches. The first one, the Genetic algorithms are based on some fact observed in (evolutionary) biology, while the second one, the Simulated annealing uses a method comes from solid-state physics.

3.1 Genetic Algorithms

Genetic algorithms are a particular class of evolutionary algorithms [9, 11, 1]. They use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and recombination. The possible states and so, the possible solutions are represented by sequences of chromosomes (genomes), they are called instances. The evolution usually starts from a population of randomly generated instances. So, a set of states, the so-called population is considered in each time, these sets form the generations. Some of the instances are chosen from a generation based on a fitness function to make the new (next) generation. To try to get better instances (better states that are closer to the ideal solution) various operators are used. In recombination the new instance usually build up as a mixture of two chosen instances. The mutation works only on 1 instance to try a 'nearby' instance. The new instances form a new population. The new population is then used in the next iteration of the algorithm. After some iterations the algorithm stops. Finally the best obtained instances give the (relatively good) solution.

3.2 Simulated Annealing

Using Simulated annealing in computations is started by the paper [5], where the method is described in first time. This approach has a root from physics and metallurgy. A thermodynamical-like process is simulated in which a parameter, the temperature is decreasing (cooling). This parameter gives the possibility to jump to a random ‘nearby’ state. Higher temperature allows jumping to states which can be further from the optimal than the actual one. In lower temperature the jump goes to states which have higher value (closer to the optimal solution). Finally the process sticks in a state. In practice to have a relatively good solution the process is used in several times and finally the best found result gives the answer.

4 Various Concepts of Parallelism

Abstracting from the specific problem solving approaches we can identify two essentially different notions of parallelism [6]. We classify them in this section.

In the ‘and-parallelism’ (which is somehow related to the concept ‘divide and conquer’) the results of several computation branches are needed to have a solution of the problem. These branches provide some subresults and these subresults build up the desired result.

The ‘or-parallelism’ (which can be related to the so-called ‘Chinese army’ model of parallel problem solving) is the following. The parallel branches are independently trying to solve the problem. Any of them can produce the solution. There are independent computations and they are performed in parallel. Note that this notion is closely related to non-determinism in the usual sense. This concept of parallelism corresponds to consider all possible ways of a non-deterministic algorithm to run at the same time.

Table 1 shows a brief comparison of these types of parallelism.

and-parallelism	or-parallelism
divide the problem to independent subproblems	brute-force, several attempts in the same time
parallelism inside the production of the solution	any attempts may produce a/the solution
the solution is provided in a parallel way	the solution is provided in a sequential way if an attempt is successful
the solution is obtained by ‘and’ of the (sub)results of the parallel branches	the solution is obtained by ‘or’ of the results of the parallel branches
the construction of the solution is faster than sequentially	the construction of the solution goes in the same time as sequentially if one find the right guess

Table 1
A brief comparison of the two main concepts of parallelism

4.1 Possibility of Parallelism in Traditional AI Algorithms

One can imagine an ideal parallel hardware which allows to run any number of branches of a program in the same time.

To run Backtracking search on this hardware would be very efficient in the following way. At each branching node of tree 1 less new branches would continue the search as many the branching factor of the node. In this case the tree form of the state-space would be traversaled by levels in a parallel way. Using this or-parallelism the algorithm does not need any backtracking.

The work of this algorithm is very similar to the ideal parallel work of the Breadth-first search. The only difference is the following: while Backtracking search uses the tree form of the state-space graph, the Breadth-first search strategy uses the original graph, therefore it does not visit the same state more than once. In this way the Breadth-first search usually (i.e. in the case when the state space graph is not tree) needs less parallel branches than the Backtracking search algorithm.

So, extending these algorithms in a parallel way the or-parallelism will be used. This extension can be on a computer with more than 1 CPU, or on a GRID-cluster. Using GRID in computation there is not trivial to check whether the given state can be and have been obtained in a different way as well. We note, here that on GRID computations usually or-parallelism is used without checking each time if the given state is already visited by another machine. GRIDs provide the ability to perform computations on large data sets, by breaking them down into many smaller ones, or provide the ability to perform several (independent) computation steps in the same time, by modelling a parallel division of labour between processes [3].

The problem with state-space represented in Figures 1 and 2 can be solved in a parallel way as Table 2 shows. After the second step the last branch finds the solution.

START	START	START	START	START	START
1	1	9	9	11	12
2	8	8	10	2	T

Table 2
Finding the terminal node and so the solution in a parallel way

Now let us see what kind of parallelism can be present in the Problem-reduction approach if one can use a parallel machine. The subproblems of the problem can be handled independently of each other. They can be solved in a parallel way in the same time. The solution of these subproblems altogether will give the solution of the main problem. At the example of tower of Hanoi a process which try to solve a subproblem, i.e. a smaller task does not depend on any other tasks. When all tasks have finished their work, the concatenation of their answers gives the answer of the original problem. This is exactly the case of and-parallelism.

4.2 Parallelism in CI Approaches

In this subsection we are analysing the used and the possible parallelism in some kinds of CI algorithms recalled earlier.

For the first sight Genetic algorithms use or-parallelism. The instances represent states of the state-space and any of them can be a candidate of the desired solution. But the instances develop in a non-independent way. The operations crossover, recombination can mix and underlie the desired properties of the given instances. So we can say that this kind of parallelism is a mixture of the clear and-parallelism and the clear or-parallelism.

In Simulated annealing usually 1 branch of computing is running in a time-point. That is a sequential idea of problem solving. Opposite to this fact, to get a relatively good solution the algorithm is used several times (in a traditional computer one after one). These runs can go in a parallel independent way if the hardware allows it. In this way the clear form of the or-parallelism is used.

Conclusions

Parallel computing is one of the main tasks of Information Technology and Computer Science [2]. There are several programming techniques to use parallel hardware. To have a more efficient way of problem solving the concept ‘parallelism’ and its possible notions are analysed. In this paper two main concepts of the parallelism are presented. They are the 'and'- and 'or'-parallelism. Their appearance in artificial intelligence and computational intelligence algorithms are analysed. We have seen that Backtracking search algorithm is entirely sequential. The Breadth-first graph-search algorithm can be seen as an algorithm trying or-parallelism. Opposite to this the Problem-reduction approach can be seen as the main example of using and-parallelism. In Genetic algorithms a kind of mixture of these concepts appears, while in Simulated annealing or-parallelism can be used.

Acknowledgement

The author thanks the financial support of the Öveges Programme (from NKTH Nemzeti Kutatási és Technológiai Hivatal A projekt a Nemzeti Kutatási és Technológiai Hivatal támogatásával valósult meg., and KPI) and of the Hungarian Ministry of Education (OM). The project is partly supported by OTKA Grants F043090 and T049409.

References

[1] Attila Álmos, Sándor Győri, Gábor Horváth, Annamária Várkonyiné Kóczy: Genetikus algoritmusok, Typotex, Budapest, 2002 (Genetic Algorithms, in Hungarian)

[2] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar: Introduction to Parallel Computing: Design and Analysis of Algorithms, Addison-Wesley, 2003

[3] Grid Computing Information Centre at http://www.gridcomputing.com/

[4] Jozef Kelemen, Sára Nagy: Bevezetés a mesterséges intelligencia elméletébe, Tankönyvkiadó, Budapest, 1991 (ELTE Mesterséges intelligencia sorozat) (Introduction to Arificial Intelligence, in Hungarian)

[5] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi: Optimization by Simulated Annealing, Science, Vol. 220, No. 4598, pp. 671-680, 1983

[6] Remco Loos, Benedek Nagy: On the Concepts of Parallelism in Biomolecular Computing, presented in research seminar of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, and submitted to publication

[7] Elaine Rich, Kevin Knight: Artificial Intelligence, McGraw-Hill, New York, 1991

[8] Stuart Russel, Peter Norvig: Artificial Intelligence. A modern Approach. Pearson Education Inc. – Prentice Hall 2003 (Hungarian translation: Panem 2005)

[9] Hans-Paul Schwefel, Ingo Wegener, Klaus Weinert: Advances in Computational Intelligence. Theory and Practice. Springer, 2003 (Natural Computing Series)

[10] Steven S. Skiena: The Algorithm Design Manual, Springer-Verlag, New York, 1997

[11] William M. Spears: Evolutionary Algorithms. The Role of Mutation and Recombination. Springer, Berlin, Heidelberg, New York, 2000 (Natural Computing Series)

[12] D. H. Wolpert, W. G. Macready: No Free Lunch Theorems for Search, Technical Report SFI-TR-95-02-010 (Santa Fe Institute), 1995