Experimenting with IDA* Search Algorithm in Heterogeneous Pervasive Environments

Stan Kurkovsky

Department of Computer Science, Connecticut State University
1615 Stanley Street, New Britain, CT 0050
kurkovskysta@ccsu.edu

Abstract. Today, mobile and smart phones are often viewed as enablers of pervasive computing systems because they provide anytime and anywhere access to information services and computational resources. However, mobile devices are inherently constrained in their computational power and battery capacity making them mere “dumb terminals” connected to a resource-rich pervasive environment. If they are ever to play a more prominent role as true elements of a pervasive environment, mobile devices must be able to embed more application logic and delegate processing requests to pervasive infrastructure. In this paper we discuss distribution and offloading of computationally intensive tasks in pervasive environments populated by mobile devices. This approach is illustrated by experimenting with a distributed version of iterative deepening A* search algorithm. In our approach, the solution space of a problem being solved is partitioned and distributed among heterogeneous mobile devices, which yields a significant increase in the time of finding an optimal solution. Distributed IDA* search algorithm does not require any coordination or communication between mobile devices, but added inter-processor communication through shared memory further increases the efficiency of the algorithm. This paper presents the results of our experiments with the algorithm and discusses a number of issues related to its implementation.

1 Introduction

A simple idea was introduced by Mark Weiser in his seminal paper [13] published over 15 years ago: as digital technology advances, computing equipment will become increasingly smaller but more powerful, which would allow ubiquitous devices to be invisibly embedded in the surrounding environment. By providing seamless communication, these miniature devices and their support infrastructure will be able to enable a computing system that is omnipresent and easy to use. Today, this idea is referred to as pervasive computing – an environment where access to computational an information processing power is available anytime and anywhere.

There are many open questions in pervasive computing research. One of them is how to create a computing system that is invisibly and seamlessly embedded in the user surroundings while minimizing the user’s perception of the system’s possibly intrusive actions? There are two mutually complementing approaches to creating pervasive systems that are unobtrusively embedded into the environment: device miniaturization and distribution of logic and computational capabilities, and achieving a level of the system's intelligence that will be able to anticipate user's actions in a given context. As a result, pervasive systems are envisioned to "fade into the background" when the users will be able to interact with these systems as naturally as they would with each other, without giving much thought to this process and focusing more on the tasks at hand rather than on the idiosyncrasies of the system's interface.

Today, in many respects, pervasive computing systems are fact rather than fiction or bold vision of the future. In particular, a fundamental property of pervasive systems is integration with the environment and providing ubiquitous services to the users. Smart phones and other network-enabled mobile devices (e.g. iPod Touch) can be viewed as a key element providing the user interface to the pervasive computing environment. Anytime and anywhere availability of a pervasive computing system is offered by the very nature of mobile phones – they are designed to provide network access at any place whenever it is needed. Today, mobile phones can act as personalized entry points for many networked information services. There is a large number of commercial and academic projects that offer a range of services which include mapping and navigation applications, personalized shopping list management, and media acquisition and streaming. Personalization, a necessary component of many pervasive services, is typically enabled by implementing different approaches to context-awareness that vary from elementary user preferences to complex profile management and matching.

Despite all the advantages of using ubiquitous mobile devices as gateways to pervasive computing systems, they suffer from one inherent problem that is rooted in their miniature size. Many modern mobile devices a significantly constrained in the amount of built-in resources, in particular, processor power and battery capacity, both of which may be a significant limiting factor in solving computationally intensive tasks. A solution to this problem could be found by exploiting the intrinsic networking features of a pervasive computing environment. A computationally intensive task could be offloaded from the constrained device to the pervasive infrastructure and/or distributed among other devices connected o the system. Here, we aim to illustrate the feasibility of this approach.

In this paper, we discuss a distributed version of IDA* search algorithm for solving computational problems in a pervasive computing environment consisting of heterogeneous mobile devices. The nature of such problems and the network infrastructure restricts the implementation of any distributed algorithm due to the limitations of the network bandwidth and computational resources available to device. Proposed distributed implementation of IDA* search algorithm is suitable for a wide array of distributed computing applications hat can be solved in such environments. We consider a community of mobile devices from the distributed artificial intelligence perspective, which views a pervasive system as a multi-agent system [9]. Computational nodes are viewed as user-centric, autonomous, intelligent agents capable of performing their own tasks, sharing their resources and communicating with other agents within the system [8].

This paper is organized as follows. Section 2 briefly discusses the necessary background related to intelligent agents, multi-agent systems and heterogeneous pervasive computing systems. Section 3 presents our distributed implementation of IDA* algorithm. Section 4 focuses on the issues related to increasing the performance and robustness of the algorithm, while Section 5 describes the experimental results. Section 6 concludes the paper.

2 Background

Intelligent agents are interactive entities that exist as a part of an environment shared with other agents capable of communicating and cooperating with one another. A multi-agent system is a loosely coupled network of intelligent agents working together to solve problems that cannot be solved by any of the individual agents due to their limited individual capabilities [1]. The term multi-agent system can also be used to describe all types of systems composed of multiple autonomous components displaying the following characteristics [5]:

- None of the agents have complete capabilities to solve the entire problem,
- There is no global system control,
- Available data is decentralized among many agents, and
- There is no centralized synchronization of computations.

In this work we use all these characteristics as the general design principles that can be applied to pervasive computing systems populated by mobile devices (computational nodes) and problem-solving algorithms that they use. Pervasive systems and applications considered here usually have the following characteristics: processing power of each mobile device within the pervasive computing system is relatively low; at least one of these devices is designated as a *coordinator* that distributes computational tasks to other computational nodes; there is no direct communication among the nodes of the system; and the size of the computational problem is exceedingly large. At the same time, mobile devices or *computing nodes* of a pervasive computing system may be viewed as individual agents within a multi-agent system: they are capable of solving their own tasks, but larger computational problems require cooperation with other agents; each computational node has access to its own data; and computations of each node are largely unsynchronized with the others.

3 Distributed Implementation of IDA* Search Algorithm

Search algorithms provide a universal mechanism for solving problems in artificial intelligence and many other areas of computer science, mathematics and engineering. In particular, different search algorithms can be used to solve discrete optimization problems [4]. To solve a discrete optimization problem, an objective function depending on one or more discrete variables must be minimized. Most discrete optimization problems are NP-complete and their state space and solution time grows exponentially. Applying a parallel or a distributed algorithm to solve a discrete optimization problem cannot reduce the time of the worst-case solution without an exponentially increased number of processors. It is possible, however, to find heuristic functions and corresponding algorithms for specific problems, which may help to reduce the average solution time to polynomial. Such an approach may be especially important for problems where real-time solutions are needed. Such heuristic-based algorithms may be applied to find a sub-optimal solution that may be acceptable in certain problem domains [7].

Many discrete optimization problems may be viewed as the problem of finding a path in a state space graph from a given initial node to one or more goal nodes [10]. The quality of a solution is measured by a cost function evaluating the path corresponding to every solution. Following this approach, a discrete optimization problem may be considered as an ordered pair (S, f), where the set of feasible solutions S is a finite or countable infinite set of all solutions that satisfy a given set of constraints. The function $f : S \rightarrow R$ is the cost function that maps each element in set S onto the set of real numbers R. The objective of a discrete optimization problem is to find a feasible solution X_0, such that $f(X_0) < f(X)$ for all $X \in S$. In the vast majority of practical problems, the set of feasible solutions S is quite large, but applying heuristics may significantly reduce the time needed to find an optimal or a sub-optimal solution.

We focus on solving a discrete optimization problem using parallel iterative deepening A* (IDA*) search method. Originally, IDA* was proposed by Korf as a version of well-known A* search algorithm with linear memory requirements [6]. The IDA* search method estimates the cost of the current partial solution from initial state I to the current state n using a combined heuristic function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of solution from the initial state I to state n, and $h(n)$ is the estimated cost of the solution from state n to the goal state G.

The IDA* search algorithm reduces its memory requirements by performing a series of independent depth-first searches bounded by an *f-cost* value limiting the length of an expanded path. Therefore, each iteration of IDA* expands paths from all search space states inside the current *f-cost* contour, as shown in Fig. 1.

A number of distributed and parallel implementations of IDA* search algorithm have been proposed. A distributed version of IDA* search algorithm is implemented by Parallel Window Search, in which the entire problem search space is analyzed by each processor, but with a different value of *f-cost* [11]. There are a number of parallel implementations of IDA* search algorithm designed specifically for SIMD and MIMD architectures [12]. These implementations require a mechanism for centralized coordination of processors. Most of these algorithms are not easy to adapt to system architectures where difference between the computational power of different processors is unknown and where the capabilities for inter-processor communication are limited. Parallel implementations of IDA* search algorithm rely heavily on the ability to balance the load among different processors. In these implementations, the problem search space is partitioned into sub-problems, and each processor is assigned its own subspace to explore. For most problems, it is impossible to predict the actual size of the search space measured as the number of nodes in the search space tree that must be expanded before a solution is found. This may result in a wide disparity in the size of the problem search space assigned to different processors. To improve the speed of finding a solution, such algorithms require a complex scheme to dynamically repartition the search space to minimize processor idling without a significant increase in inter-processor communication.

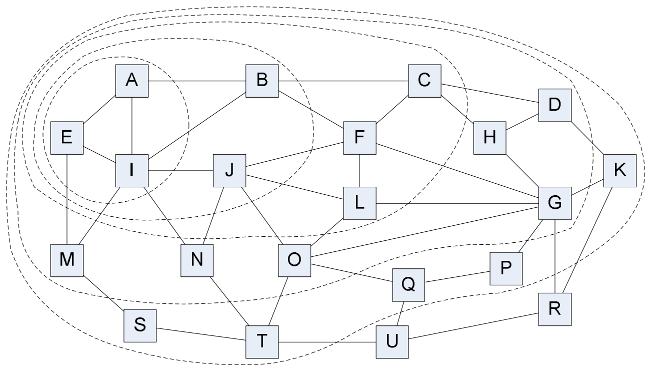

Fig. 1. A graph for a shortest path problem with *f-cost* contours

We propose a distributed implementation of IDA* algorithm designed specifically for distributed computing applications running within a pervasive computing system populated with low-powered mobile devices. Our implementation of the algorithm has the following characteristics:

- A central server is required for task initiation and solution assembly;
- Coordination of computing nodes is minimized;
- Synchronization of computation is not required;
- Communication among the computing nodes is not required;
- Computing nodes are heterogeneous and their computational power may vary; and
- Fault-tolerance is achieved by reallocating a task to a new computing node.

The core of our distributed implementation of IDA* search algorithm consists of three phases as follows:

Phase I. Search space partitioning. As shown in Fig. 2, a coordinator is required to expand several of the first levels of the search space using a standard non-distributed version of he IDA* algorithm. This generates a number of partial solutions leading from the initial node *I* to several intermediate nodes within the search space graph. Each partial solution corresponds to a subset of the solution search space, as illustrated in Fig. 2.

Phase II. Distributed search. Each partial solution generated in Phase I is assigned to a single computational node. The resulting partition of the search space corresponding to each computing node is expanded until a feasible solution is found. When the computational node finds a solution, it is optimal within the solution subspace assigned to that node. However, the node itself cannot verify the global optimality of this solution because the node cannot estimate the cost of solutions found by other nodes.

Phase III. Solution assembly. After a computational node finds a feasible solution within the search subspace assigned to it, this solution is returned to the coordinator. It is the coordinator's responsibility to identify an optimal solution among many feasible solutions based on the optimality criteria (i.e. solution cost) specific to the given problem.

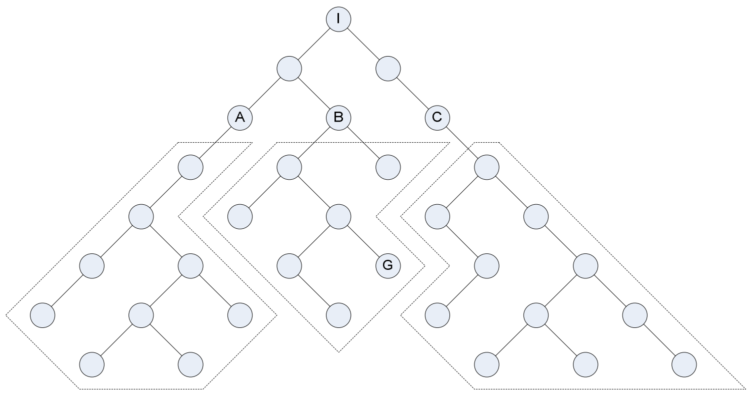

Fig. 2. Search space of distributed implementation of IDA* algorithm

None of the three phases of our distributed implementation of IDA* search algorithm require any synchronization or significant communication between each of the computing nodes and the coordinator. Such communication is only required to take place when a computing node receives its partial solution and when it reports the found feasible solution back to the coordinator.

4 Improving Robustness and Performance of the Algorithm

A number of issues directly influence the performance of our distributed implementation of the algorithm: timeout or loss of computational nodes, finding suboptimal solutions, and boosting performance of the algorithm with shared memory.

In certain problems, such as finding a complex route on a large geographical map, a suboptimal solution to the problem may be satisfactory. In the proposed algorithm, feasible solutions arrive to the coordinator in an arbitrary order. For real-time problems, when an end-user is monitoring the results of distributed search and is waiting for a solution, the search process may be terminated as soon as a feasible solution is deemed satisfactory by the human end-user.

Adding some coordination among the computing nodes through shared memory my further increase the performance of our distributed algorithm. In the most basic distributed implementation of the algorithm, each computing node is not aware of the solution process taking place at other nodes. Consider a class of problems where the solution cost function increases monotonously with the addition of each new node to the current solution. If a feasible solution *S* with the cost *f*(*S*) has been found by a given computing node, a subset of computing nodes will not be able to find an optimal solution if these nodes are currently exploring partial solutions whose cost already exceeds *f*(*S*). Therefore, distributed tasks on such computing nodes can be terminated because it is known a priori that their solutions will not be optimal. Implementing such a strategy requires that each computing node be informed about the cost of a solution, which is currently considered best by the coordinator. This information is always available to the coordinator and can be communicated to each computing node using a two-way keep-alive protocol.

The nature of the proposed distributed implementation of IDA* search algorithm is such that the length of partial solutions generated by the coordinator is increasing with the addition of computing nodes. This results in the partitioning of the original search space and in the simplification of the task of each computing node. However, this also results in situations where computing nodes can only find feasible solutions that are far from optimal; in some cases, feasible solutions may not even exist. It is important to maintain a balance between the positive and negative effects of partitioning of the search space. If the chosen number of computing nodes is too high, too many of them will be exploring "dead end" search subspaces, where only expensive feasible solutions can be found. In such situations, the coordinator's computational load will be too high because it will need to generate an excessive number of partial solutions. Coordinator also will be overloaded with a very high communication overhead because it will need to send all generated partial solutions to the computing nodes and receive solutions back from them.

The efficiency of a distributed implementation of IDA* search algorithm can be measured by the speedup gained by adding more computing nodes. Given a problem of a fixed size (as discussed in the following section) the speedup usually does not increase linearly with the number of computing nodes. This indicates that the overall efficiency of the problem-solving process eventually starts to decrease as the total number of computing nodes grows. This situation is best described by Amdahl's law stipulating that every algorithm has a certain sequential component, which cannot be reduced by adding more processors; therefore as the number of processors grows, the efficiency of the system drops.

5 Experimenting with the Algorithm

To prove the viability of our distributed implementation of IDA* search algorithm, we conducted a range of computational experiments. We considered a problem of finding the shortest path on a geographical map consisting of 180 locations. The feasibility of the algorithm was tested using a varying number of computing nodes running on actual Dell Axim PDAs and simulated mobile devices with realistic performance characteristics running within emulators on host workstations. A central server (coordinator) was responsible for generating partial solutions, sending them along with the adjacency matrix to each computing node, collecting the results from them, and identifying the optimal solution from a number of feasible solutions. In these experiments, no provisions were made for timing out of the computing nodes since the computers were used exclusively for this experiment.

Using the same adjacency matrix, we considered four different instances of the shortest path problem with different initial and goal locations. Each instance of the problem was solved using a non-distributed version of the IDA* search algorithm. The time of finding this solution was used to calculate the relative speedup achieved by the pervasive system by using distributed implementation of the same algorithm. Each instance of the algorithm was solved by the pervasive computing system consisting of an incrementally increasing number of computing nodes.

In the first series of experiments, we tested our distributed implementation of IDA* search algorithm with no shared memory. Each computing node received its own partial task together with the adjacency matrix from the coordinator and was allowed to run the algorithm until a feasible solution was found or it was determined that a feasible solution does not exist due to the restrictions imposed by the original partial solution. Upon the completion of this task, each computing node sent its results back to the coordinator. In each experiment in this series, we solved each of the four problems using a fixed number of computing nodes; the number of computing nodes gradually changed from one to fifty in the experiments conducted in this series. In our implementation of the algorithm, using just one computing node was different from solving the same problem using a non-distributed algorithm. If a distributed algorithm uses a single computing node, that node starts computations from the partial path it received, which consists of the initial location and one of its immediate neighbors. In each experiment, we measured the time needed by a computing node to complete its respective computational task. The speedup was measured as the ratio of the time needed to solve a given problem using a non-distributed IDA* algorithm and the average time taken by the entire pervasive computing system to solve the same problem. Fig. 3 shows the complete results of this series of experiments.

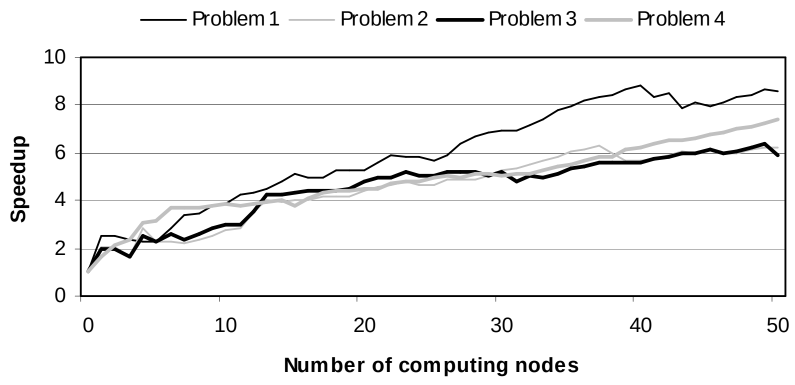

Fig. 3. Speedup achieved by distributed IDA* search algorithm with no shared memory

As can be seen in Fig. 3, the speedup tends to increase as the number of computing nodes grows. The graphs shown in Fig. 3 contain a mix of surges and plateaus with occasional dips. The surges are formed when the coordinator generates a new partial solution that significantly reduces the search space for one of the computing nodes. For example, consider a problem of finding the shortest path from *I* to *G* on a graph shown in Fig. 1. Suppose that for a given experiment, there are six computing nodes, which receive the following partial solutions: *I-A*, *I-B*, *I-E*, *I-J*, *I-M* and *I-N*. The next experiment will include seven computing nodes, which will receive the following partial solutions: *I-A*, *I-B*, *I-E*, *I-J-F*, *I-J-L*, *I-M* and *I-N*. It is possible to assume that the computing nodes receiving partial solutions *I-J-F* and *I-J-L* will produce optimal solutions significantly faster than other computing nodes because the size of their search spaces has been significantly decreased.

Given the discrete nature of the shortest path problem, certain combinations of partial solutions resulted in a better solution time compared to other, often very similar combinations. The dips in the graphs shown in Fig. 3 are formed as a result of moving from a “good” combination of partial solution to a somewhat “worse” combination. Nevertheless, the overall trend of the graphs reflects the improved speed of finding an optimal solution with the increased number of computing nodes. Obviously, the plateaus in the graphs shown in Fig. 3 are generates by sequences of experiments that do not result in a surge or a dip. It is important to note that the increasing speedup trend is not going to continue indefinitely. Eventually, the curve will saturate due to the reasons described in the previous section.

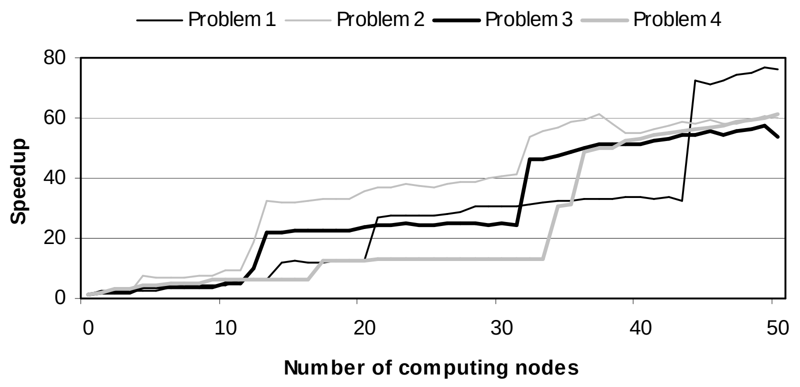

Fig. 4. Speedup achieved by distributed IDA* search algorithm with shared memory

In the second series of experiments, we tested our distributed implementation of IDA* search algorithm with shared memory (Fig. 4). Similarly to the first series of experiments, each computing node received its own partial task and the corresponding adjacency matrix from the coordinator. Unlike the previous series of experiments, each computing node periodically received the cost value *f*(*S*) of a feasible solution *S* that was currently considered best by the coordinator. This information was delivered using the keep-alive protocol discussed in the previous section. In a situation when a given computing node *N* is working on a solution S_n whose current cost $f(S_n) \geq f(S)$, *N* was required to terminate its calculations because it would be unable to find a feasible solution whose cost is lower than $f(S)$. The results of measuring the speedup achieved by our distributed implementation of IDA* algorithm with shared memory are shown in Fig. 4. Overall, the speedup is several times higher than in the previous series of experiments, in which we tested our implementation without using shared memory. It is also noteworthy that the speedup graphs of this experiment tend to have more pronounced surges and very few dips. Although the performance of this implementation is significantly better than of the previous one, it is achieved by a periodic exchange of messages between the coordinator and each of the computing nodes using the keep-alive protocol. It is also important to note that the communication overhead increases with the number of computing nodes. Similarly to the previous series of experiments, a very high number of nodes may result in the overloading of the coordinator and in the overall decrease in the efficiency of the problem-solving process.

6 Summary

In this paper we presented a distributed implementation of IDA* search algorithm. This algorithm is designed specifically for distributed applications running within a pervasive computing system and is aimed to illustrate the feasibility of a mechanism for offloading computationally intensive tasks from a mobile device with limited resources and distributing these tasks within a pervasive system. Such pervasive systems consist of a large number of mobile devices with relatively low-power processors. Computational problems solved in such systems are initiated by a single computational node, which also serves a coordinator in the problem-solving process. We view the community of computational nodes as a multi-agent system, in which each node is capable of solving its own task, but it has limited resources and therefore must rely on cooperation with other nodes to solve larger tasks.

Experimental results presented in this paper indicate that distributed implementation of IDA* search algorithm can be enhanced by allowing computational nodes to communicate via a shared memory. However, it is important to keep the balance between the gains in speed and the overhead of exchanging messages needed to implement the shared memory.

References

1. Durfee, E., Lesser, V., Corkill, D.: Trends in Cooperative Distributed Problem Solving. IEEE Transactions on Knowledge and Data Engineering, March 1989, 1(1), pp. 63-83.
2. Folding@home, Protein Folding Simulation Project: Available at http://folding.stanford.edu.

3. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 15(3), 2001.
4. Grama, A., Kumar, V.: State of the Art in Parallel Search Techniques for Discrete Optimization Problems. IEEE Transactions on Knowledge and Data Engineering. January/February 1999, pp. 28-35.
5. Jennings, N., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems Journal, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston, 1998, 1(1), pp. 7-38.
6. Korf, R.: Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 27(1985), pp. 97-109.
7. Korf, R.: Artificial Intelligence Search Algorithms. Algorithms and Theory of Computation Handbook, CRC Press, 1999.
8. Kuang, H., Bic, L., Dillencourt, M.: Iterative Grid-Based Computing Using Mobile Agents. In Proceedings of the 2002 International Conference on Parallel Processing, T. Abdelrahman (Ed.), 2002, pp. 109-117.
9. O'Hare, G., Jennings, N.: Foundations of Distributed Artificial Intelligence, John Wiley and Sons, 1996.
10. Pearl, J.: Heuristic-Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.
11. Powley, C., Korf, R.: Single-agent Parallel Window Search. IEEE Transactions on Pattern Analysis. 13(5), 1991, pp. 466-477.
12. Reinefeld, A., Schnecke, V.: AIDA* – Asynchronous Parallel IDA*. In Proceedings of 10th Canadian Conference on Artificial Intelligence, Banff, Alberta, 1994, pp. 295-302.
13. M. Weiser, “The Computer for the Twenty-First Century,” Scientific American, Sep 1991, pp. 94-104.