Agent-Based Distributed IDA* Search Algorithm for a Grid of Mobile Devices

Stanislav KURKOVSKY
and
BHAGYAVATI
Department of Computer Science, Columbus State University
Columbus, GA 31907, USA

ABSTRACT

We present a distributed version of IDA search algorithm for a computational grid. We also propose a grid architecture based on a multi-agent system paradigm applied to mobile devices on a cellular network. The proposed algorithm is specifically designed to adapt to the flexible environment of the grid. In this paper, we discuss several aspects of the distributed IDA* algorithm and the underlying grid architecture including distributed task partitioning and distribution, assembly of an optimal solution, agent activities, mobility issues, network configuration, and agent communication protocols. We conclude with the experimental design to test the grid architecture and our implementation of the IDA* algorithm.*

Keywords: Distributed IDA* search algorithm, Multi-agent systems, Computational grids, Cellular networks.

1. INTRODUCTION

Distributed problem-solving algorithms play a major role in enabling computational grid applications. Grid applications aggregate resources of many computing devices on a wired or wireless network to solve a single problem at the same time. In most applications, computational grids are considered in the context of sharing the processor power of many computers interconnected by a TCP/IP (Transmission Control Protocol/Internet Protocol) based network. Popular examples of grid computing applications are the Genome@home Human Genome Project [5], Folding@home simulation of protein folding and aggregation [3], Search for Extraterrestrial Intelligence (SETI@HOME) project [16] and The Great Internet Mersenne Prime Search (GIMPS) project [6]. In all of these projects, thousands of users are sharing unused CPU cycles of their computers for performing computationally-expensive processing. The Globus Project [4] is the current de facto standard for large-scale grid applications. The Globus Toolkit provides a software library for enabling grid computing for solving scientific and engineering problems.

In this paper, we propose a distributed version of IDA* search algorithm for a computational grid of mobile devices. As we will show, the nature of such a grid imposes significant restrictions on the implementation of any distributed algorithm due to the limitations on communication channel capacity and computational resources available to each device on the grid. Our implementation of distributed IDA* search algorithm can be used in many applications typically solved by handheld and mobile devices; in particular, vehicle navigation or travel routing problem.

We focus on heterogeneous computational grids comprised of mobile and wireless devices, such as smart cellular telephones and PDA's. Due to the nature of these devices, we consider a computational grid from the perspective of Distributed Artificial Intelligence [12], which views a computational grid as a multi-agent system. Computational devices on the grid are user-centric, autonomous, intelligent agents capable of performing their own tasks, sharing their resources and communicating with other agents on the grid [11, 17]. Specifically, we explore a kind of computational grid that consists of mobile devices on a cellular network. Our motivation for considering a computational grid consisting of mobile devices in cellular networks is explained next.

Cellular wireless networks are more constrained than traditional wired networks because the bandwidth is strictly limited. On such a network, typically, there are many users competing for the scarce, available spectrum. Furthermore, the small size of the mobile device leads to limited processing power. Mobile devices in our consideration generally do not have adequate amounts of memory for computationally-intensive tasks. Since mobile devices communicating through wireless channels characteristically have fewer resources than their wired counterparts, a computational grid proves to be a very useful architecture to solve the vexing problem of limited resource devices performing resource-intensive tasks.

2. BACKGROUND

In this section we briefly describe the necessary background for understanding multi-agent systems, cellular networks and computational grids.

Intelligent agents are interactive entities that exist as

part of an environment shared with other agents capable of communicating and cooperating with one another [13]. 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 [2]. The term multi-agent system can also be used to describe all types of systems composed of multiple autonomous components displaying the following characteristics [8]:

- Each agent has incomplete capabilities to solve a problem,
- There is no global system control
- Data are decentralized, and
- Computations are asynchronous.

Later in this paper, we tie these requirements with our proposed architecture.

In our work, we only consider cellular networks and not other kinds of wireless and mobile networks such as wireless local area networks and wireless wide area networks. In a cellular network, the geographical area of service is divided into cells. Each cell has a base station that uses wireless transmission technologies to provide services to mobile users in its area [1]. When a mobile device moves from one cell to another, mechanisms are needed to ensure continuity in communication. Such mechanisms are called handoff mechanisms. In order to increase capacity to meet the ever-increasing demand for wireless services, cellular service providers divide a cell into logical sub-cells. These smaller and more numerous cells allow for lower power consumption per cell along with lower transmission power. Therefore, more customers can now be serviced than if the cell were not split.

Current mobile devices provide end-users with a basic level of ubiquitous computing. Ubiquitous computing involves the establishment of a computing system that is unobtrusively embedded in the environment, completely connected and constantly available. Devices providing such pervasive computing are typically not the wired desktop machines that are dominant today, but are compact mobile or embedded devices communicating through hybrid wired-wireless networks. In order to address the aforementioned issue of diminished resources in mobile devices, we turn to grid computing, which concurrently applies the computing resources in the network to a single, intensive problem.

3. DISTRIBUTED IDA* SEARCH ALGORITHM

Search algorithms provide a universal problem-solving mechanism in artificial intelligence and other related areas of computer science. Discrete optimization problems (DOP) form a class of problems that can be successfully solved using various search algorithms [7]. To solve a DOP, one needs to minimize an objective function of a set of variables that can only have discrete values. Most DOP's are NP complete and their solution time grows exponentially. Applying a parallel algorithm to solve a DOP cannot reduce the worst case run-time without exponentially increasing the number of processors. However, it is possible to find heuristic algorithms for specific problems that would help reduce the average run time to polynomial time, which is especially important in situations when a solution must be obtained in real time. Such heuristic algorithms may also be used to quickly find sub-optimal solutions, which may be acceptable for certain problems [10].

Some instances of DOP's can 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. 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 can be viewed 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 DOP 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 will illustrate our discussion of IDA* search algorithm by solving a vehicle routing problem. A task of this nature and scale is a good fit for mobile devices under consideration, such as smart cellular telephones, PDA's, tablet PC's or mobile laptops, which are routinely used for routing on metropolitan or regional area maps. An instance of the shortest (or quickest) path routing problem expected to be solved by mobile devices on a grid is formulated as follows. Given a geographical map showing cities, and roads connecting them, we need to find the shortest path from city I to city G. Distances between certain pairs of cities are known. If the corresponding graph does not show a connection between a pair of cities, there is no direct path between them.

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 [9]. The IDA* search method estimates the length of the shortest path through city n using a combined heuristic function $f(n) = g(n) + h(n)$, where $g(n)$ is the length of path from the initial city I to city n, and $h(n)$ is the estimated cost of the shortest path from city n to the goal city G.

The IDA* search method 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 cities inside the current *f-cost* contour, as shown in Figure 1.

```
function IDA*(problem) → solution
  loop
    solution, f-limit ← Contour(root, f-limit)
    if solution <> null then return solution
    if f-limit = ∞ then return failure
  end loop
end function

function Contour(node, f-limit) →
              solution, new f-cost limit
  if f-cost(node) > f-limit then
    return null, f-limit
  if node = goal then return node, f-limit
  next-f ← ∞
  for each node n in Children(node) do
    solution, new-f ← Contour(n, f-limit)
    if solution <> null then
      return solution, f-limit
    next-f ← min(next-f, new-f)
  end for
  return null, next-f
end function
```

Figure 1. Pseudo Code for IDA* Search Algorithm

There are several known distributed implementations of IDA* search algorithm. Parallel Window Search implements a distributed version of IDA* where each processor examines the entire search space with a different level of *f-cost* [14]. There are other parallel implementations of IDA* specific to SIMD or MIMD architectures [15]. However, these algorithms require centralized coordination and are not easily adaptable to architectures in which inter-processor communication is limited and processors may vary in power. We propose a distributed implementation of IDA* algorithm for a computational grid of mobile devices with the following characteristics:

- The algorithm requires minimal coordination;
- Inter-processor communication is not required (except for very limited communication with the coordinator);
- Processors (mobile devices) may have varying resources available to them;
- The algorithm is fault-tolerant and can easily recover from losing one or more processors.

In our approach, the IDA* algorithm can be modified to run in parallel on several mobile devices. Our parallel implementation of the IDA* search method for a computational grid consists of the following three phases:

Initial data partitioning phase. As shown in Figure 2, a mobile device initiating the problem-solving process expands the first few search tree levels using the standard iterative deepening algorithm until it generates a sufficient amount of search tree nodes. Then it submits these nodes, which are partial solutions, or paths leading to these nodes, to the coordinator and requests that subsequent search phases be solved by other devices on the grid.

Distributed search phase. Each mobile device that joined the process of solving this distributed task explores its own search subtrees received from the coordinator until a solution is found. The coordinator is responsible for choosing a mobile device that would have enough resources to complete the partial task assigned to it. If a given sub-task exceeds the computational power of a device to which it is assigned, this device can act as an initiator of its own partial task.

Solution assembly phase. When a mobile device working on a problem finds the solution to its partial task, the solution may not be optimal because it was bounded by the partial path received from the coordinator. When the mobile device that initiated the original task receives a set of completed partial solutions from the coordinator, it needs to select the best one corresponding to a path with the shortest length.

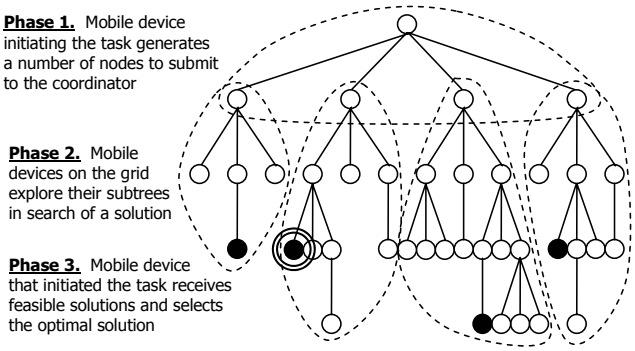

Figure 2. Distributed IDA* and its Search Space

None of the three phases of the parallel IDA* algorithm require any significant synchronization or communication between each device and the coordinator. The only two occasions that communication takes place between mobile devices and coordinator are when the devices receive their portions of a distributed task, and when they submit their partial solutions to the coordinator.

4. AGENT-BASED GRID ARCHITECTURE

A distributed implementation of the IDA* search algorithm described above requires that each mobile device participating in solving a distributed problem on the grid have a high degree of autonomy and be able to communicate with the coordinator using a well-defined communication interface. Using a multi-agent system approach in which all mobile devices are viewed as intelligent agents, we propose a cellular network-based grid architecture that consists of the following elements:

- Initiator – a mobile device that requests and initiates the process of solving a distributed task;
- Subordinates – all other mobile devices available on the grid to solve a distributed task;
- Brokering Service – a part of the grid infrastructure that coordinates the distributed problem-solving process.

All mobile devices currently available in the wireless cell advertise themselves as available by polling the

Brokering Service provided by the wireless network infrastructure. The responsibility of the Brokering Service is to facilitate communication among mobile devices on the grid and to coordinate their work on distributed tasks. A mobile device referred to as the Initiator may initiate a task by announcing it through the Brokering Service. All other devices referred to as the Subordinates may participate in the task if they are not currently busy with other tasks or report their solutions to the Brokering Service if they have completed their assigned sub-tasks. The Initiator assembles the complete solution out of the partial solutions received through the Brokering Service.

4.1 Components of the Computational Grid Based on Mobile Devices

The medium of the computational grid of our research is a wireless cell. The primary aspect of the cellular network that we must consider is that the devices on the grid are mobile and they may introduce a high degree of instability into the network. Owners of mobile devices such as telephones and PDA's may leave or enter the given cell at any moment of time; we cannot make any assumptions about how long each mobile device will stay within the current cell and, therefore, we cannot guarantee that each mobile device will successfully finish working on its share of the distributed computational task before leaving the cell.

A typical computational grid unites a number of devices that can share their processing power and/or storage to work on solving a shared problem in parallel. We view each such device as an intelligent agent that represents a smart cellular telephone, a wireless-enabled PDA, a wireless-enabled laptop, or a tablet PC with wireless network access. In our architecture, agents do not and cannot communicate with one another directly; instead, they communicate through the Brokering Service provided. For example, we disregard the ability of many PDA's to transfer data directly among one another through an infrared port as irrelevant to our proposed architecture.

Each agent entering the cell must register itself with and inform the network infrastructure about its characteristics, such as computational power, amount of available resources and types of tasks that it is capable of solving. Knowledge of the computational power and the amount of resources available to each agent will help the Brokering Service decide how to allocate portions of distributed computational tasks to each agent.

Intelligent agents must periodically inform the Brokering Service about their presence within the cell, their current workload and/or progress towards obtaining a solution. The agents use the keep-alive protocol, which is discussed below, to send such information. We also recognize that a local task has a higher priority for a mobile device than the distributed task. Solving a local task such as placing a call or searching for a record in its address book should have a higher priority for a Subordinate than participating in an Initiator's task.

The Brokering Service provided by the grid infrastructure has several related responsibilities. First, it must keep an up-to-date Active Agent Repository of all intelligent agents available within the given cell along with their current workload, computational abilities such as CPU power, and the code libraries available to it. Second, it must assign the sub-tasks received by the Initiator to available and willing Subordinates. The Brokering Service knows about Subordinate agents' availability through the Active Agent Repository. Third, the Brokering Service keeps track of all partial tasks in its Task Allocation tables. Each partial task is marked as "unassigned", "assigned" or "completed." Finally, after completing their assigned sub-tasks, Subordinates return the partial results to the Brokering Service, which is then responsible for caching the results until they are retrieved by the Initiator. After the results have been retrieved by the Initiator, the Brokering Service deletes the corresponding partial tasks from its Task Allocation tables.

Any agent on the grid can act as an Initiator of a distributed task. If an agent has a task that it cannot solve by itself and the task is such that it can be effectively distributed across a computational grid, the agent can become an Initiator and announce the task to other agents on the grid via the Brokering Service. If there are other intelligent agents in the cell capable of solving such a distributed task and are able to communicate their characteristics, the Brokering Service will facilitate distribution and resolution of this task. Like any other agent, the Initiator must poll the Brokering Service through the keep-alive protocol, described in a later section, and retrieve any partial solutions that may have already been submitted by Subordinates. It is the Initiator's responsibility to split the original task into sub-tasks distributable by the Brokering Service. The Initiator is also responsible for assembling the complete solution to the distributed task after partial results are returned to the Brokering Service by the Subordinates.

Any intelligent agent that does not serve as an Initiator of a distributed task at a given moment of time and has registered itself with the Brokering Service is considered a Subordinate. Any new agent entering a cell is considered a Subordinate. There are no restrictions for a Subordinate to become an Initiator. Therefore, it is possible for several Initiators to co-exist on a single grid in which many agents simultaneously work on several distributed tasks. Distributed tasks are assumed to have a lower priority than local tasks running on a Subordinate agent. Typically, a local task is the cause for an agent to become an Initiator. If an agent becomes an Initiator of a new distributed task while serving as a Subordinate for another Initiator, the previous distributed task is aborted to free up the agent's resources to work on its own task.

4.2 Communication Aspects

Apart from their primary functionality as smart cellular telephones, wireless PDA's, mobile laptops or wireless-enabled tablet PC's, mobile devices that we view as intelligent agents can send and receive a wide variety of messages to and from the Brokering Service in order to participate in grid activities. The following sections describe the main types of messages that are used to communicate between the Brokering Service and intelligent agents on its grid.

Join the Grid. For an intelligent agent, the process of joining a grid is associated with the process of entering a cell by a mobile device. As soon as it enters the cell, it automatically joins the grid and registers itself with the Brokering Service that it is available to solve distributed tasks. Agents cannot become Initiators if they are not registered with the Brokering Service as potential Subordinates. This prevents agents from becoming "greedy," that is, only initiating distributed tasks of their own, but refusing to potentially serve as Subordinates. Upon joining the grid, any agent that is not busy with other local or distributed tasks may be asked by the Brokering Service to become a Subordinate. In our architecture, such an agent has to honor the request.

Keep-Alive Protocol. Regardless of the current status as Initiator or Subordinate, each agent must inform the Brokering Service about its status at periodic intervals, so that its status in the Active Agent Repository is always kept current. This information should include the agent's availability to accept partial tasks as a Subordinate. When a Subordinate is working on a partial task, it must periodically confirm with the Brokering Service that it is currently occupied with an unsolved task. If the keep-alive protocol times out with a given agent, the agent is removed from the Active Agent Repository maintained by the Brokering Service. If the agent was working on a partial task, this task will be reassigned to a different Subordinate. If a keep-alive protocol times out for an Initiator, the corresponding distributed task is terminated and all Subordinates working on it are sent a message by the Brokering Service to terminate their respective partial tasks. The Brokering Service also deletes the task entries from its Task Allocation table.

Initiate a Distributed Task. Intelligent agents cannot initiate distributed tasks by themselves. There must be a local task started by the device's user that cannot be solved by the device itself and therefore warrants an initiation of a distributed task. If a local task is requested while the agent is working on a distributed task as a Subordinate, the distributed task is terminated. Therefore, any agent not currently working as a Subordinate on a partial task may request to become an Initiator if necessitated by a complex local task. If such a request is granted, the Initiator splits its task into sub-tasks and submits them to the Brokering Service. Initially, all partial tasks are marked as "unassigned" in the Task Allocation table. This status is changed as sub-tasks are allocated to Subordinates and partial results are retrieved.

Terminate a Distributed Task. If an Initiator decides to terminate the distributed task it has started due possibly to user cancellation, it sends a corresponding message to the Brokering Service, which, in turn, discards any cached partial results, clears the Task Allocation table for the given distributed task, and requests all Subordinate agents working on this distributed task to abort execution of their partial tasks.

Retrieve Partial Results (Initiator Only). After an Initiator has started a distributed task, it periodically queries the Brokering Service regarding the availability of any partial results. The Brokering Service informs the Initiator about the current progress of the distributed task by communicating one of the following:

1. **Solved.** These results have already been retrieved by the Initiator;
2. **Completed.** These distributed tasks have already been solved, but the results are awaiting retrieval by the Initiator, upon which the Brokering Service discards them and deletes corresponding partial tasks from the Task Allocation Table;
3. **Assigned.** Unsolved partial tasks have been assigned to Subordinates;
4. **Unassigned.** Partial distributed tasks that have not yet been assigned to Subordinate agents.

In order to find an optimal solution, the Initiator must assemble the partial results received from all Subordinates. However, the user who initiated the search for the shortest path may be satisfied with a "good enough" solution and decide not to wait for the optimal path. In that case, the Initiator will send a message to the Brokering Service to terminate the distributed task because partial results have already been retrieved to the user's satisfaction.

Receive a Partial Task (Subordinate Only). When the Brokering Service has an unsolved partial task, it searches for an available Subordinate agent in its Active Agent Repository. It must not only find an available agent, but also must make sure that this agent has enough resources to solve a given partial task. It ensures this by polling the Subordinate on its availability to perform that sub-task. When such a candidate has been found and has confirmed its availability, the Brokering Service sends the corresponding partial task to this agent. This partial task is marked as "assigned" in the Task Allocation table.

Submit Partial Results (Subordinate Only). After a Subordinate successfully finishes working on its partial task, it submits the results to the Brokering Service. This Subordinate may now receive new partial tasks. Newly submitted results are available for the Initiator to retrieve and the corresponding partial tasks are marked as "completed" by the Brokering Service in its Active Agent Repository. As soon as the Initiator retrieves the

partial results, it sends a message to the Brokering Service, resulting in the task being deleted from the Task Allocation table.

Terminate a Partial Task (Subordinate Only). If a Subordinate currently working on a partial task receives a local task from its user, it may have to dedicate all of its resources to solving the local task. In that case, the agent needs to terminate its partial task and inform the Brokering Services. This partial task is then marked as "unassigned" in the Active Agent repository.

5. SUMMARY AND FUTURE WORK

In this paper we presented a distributed version of IDA* search algorithm and an architecture for a computational grid comprised of mobile devices that can be used to run the algorithm. By implementing our version of IDA* algorithm on this architecture, we also intend to experimentally prove our concept of the wireless computational grid. In the immediate future, we plan to model a computational grid infrastructure based on the proposed architecture. Then we will conduct experiments of a basic nature to prove the viability of a wireless grid for mobile devices that enables sharing of scarce resources to perform a computationally-intensive task. In particular, a comparison of the time taken by a wireless-enabled PDA to obtain results using a local IDA* algorithm versus the lesser time taken by the grid to obtain similar results using our distributed version of IDA* algorithm would demonstrate its advantages.

Long-term experiments are geared toward studying the impact of different cellular network populations and device mobility on the time taken to solve a distributed task using IDA* algorithm. For example, we will measure the time taken to solve a particular task when the number of mobile devices in a cell remains relatively stable. Then we will measure the time taken to solve the same task under higher device movement conditions when Subordinates are leaving and entering the cell more rapidly than in the earlier scenario. The comparison between the two situations is intended to yield the efficiency bounds of the architecture.

Another goal is to experimentally discover the best way to distribute certain tasks. In the IDA* search algorithm discussed in this paper, if a given Subordinate agent has to expand a number of nodes in the subtree that exceeds a certain threshold proportional to the amount of resources available to this agent, it can act as an Initiator of a distributed sub-task. In this case, it submits its excess nodes with their partial solutions to the Brokering Service, which can find other agents to explore them. Our goal will be to minimize such occurrences where a Subordinate becomes an Initiator of its own partial task because the distribution of the partial tasks based on Subordinates' resource availability was not performed in an optimal fashion.

6. REFERENCES

[1] Bhagyavati. *Automated Fault Management in Wireless and Mobile Networks.* Ph.D. Dissertation, the University of Louisiana at Lafayette, Spring 2001.

[2] E. Durfee, V. Lesser and D. Corkill. Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering*, March 1989, KDE-1(1), pp. 63-83.

[3] *Folding@home – Protein Folding Simulation Project.* Available at http://folding.stanford.edu.

[4] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[5] *Genome@home – Human Genome Project.* Available at http://gah.stanford.edu.

[6] *GIMPS – The Great Internet Mersenne Prime Search*. Available at www.mersenne.org.

[7] A. Grama and V. Kumar. 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.

[8] N. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems Journal*, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston, 1998, 1(1), pages 7-38.

[9] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1985), p. 97 - 109.

[10] R. Korf. Artificial Intelligence Search Algorithms. In *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

[11] H. Kuang, L. Bic and M. Dillencourt. Iterative Grid-Based Computing Using Mobile Agents. In *Proceedings of the 2002 International Conference on Parallel Processing*, T. Abdelrahman (Ed.), 2002, pp. 109-117.

[12] G. O'Hare, N. Jennings. *Foundations of Distributed Artificial Intelligence*, John Wiley and Sons, 1996.

[13] N. Pissinou, S. Kurkovsky, R. Benton, B. Bhagyavati. A Roadmap to the Utilization of Intelligent Information Agents: Are Agents the Link Between the Database and Artificial Intelligence Communities? In *Proceedings of 1997 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX-97).*

[14] C. Powley and R. Korf. Single-agent parallel window search. IEEE Transactions on Pattern Analysis. 13(5), 1991, pp. 466-477.

[15] A. Reinefeld, V. Schnecke. AIDA* – Asynchronous Parallel IDA*. In *Proceedings of 10th Canadian Conference on Artificial Intelligence*, Banff, Alberta, 1994, pp. 295-302.

[16] *SETI @ HOME – The Search for Extraterrestrial Intelligence*. Available at http://setiathome.ssl.berkeley.edu.

[17] A. Tveit. jfipa – an Architecture for Agent-based Grid Computing, Proceedings of the *Symposium of AI and Grid Computing*, AISB Convention, 2002.