Abstract: We developed system based on JADE framework. This experimental multiagent system consists of two types of agents. Manager distributes tasks to workers, who can decide whether to accept or reject the task. The decision is made as evaluation of expression tree which takes into account agent-task compatibility and attributes of currently solved tasks. The trees are evolved by genetic algorithm and fitness function reflects time consumed for solving the whole task set and average time per one task. The shortest times are achieved for short agent’s queues. Best evolved expression tree leads to average time per task 1013 ms. If decided randomly, the most compatible task is solved in 4150 ms.

Keywords: genetics, agent, computation, multi-agent, decision making, decision trees

1 Introduction

An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives [4]. Main features of an agent are interaction with other agents and adaptivity to environment states.

Very often a complex problem could be divided into a set of subtasks. We cannot in principle control central assignment of individual subtasks to agents in dynamic computations systems or environment. For example the manager agent cannot know every local variables and conditions of all worker agents, so he cannot prepare optimal plan for task delivery. There is only a chance to optimize general criteria of complex problem by controlling computation effectively in bottom-up direction.

2 System architecture

We developed independent module to experimental Multi-Agent computational System (MAS) [3]. This system named Pikater is based on open source JADE [1] framework written in Java. The system consists of several types of agents. The first one is ”manager” agent which offers computational tasks to another type of agents – ”workers”. Manager communicates with workers, who are answering him, but they don’t communicate among themselves. Every worker agent solves tasks in his incoming buffer.

Our module now brings ”brain” to worker agents. A worker have now two options when a new task is received. In past he had to accept all incoming tasks. Now, he can either accept a task and store it into buffer or reject it. In both situations he sends a message with his decision back to manager. With this artificial brain the worker agents can make decision, based on local conditions and variables, whether to accept offered task from manager or reject it.

2.1 Computing tasks

Computation task is now an artificial dummy task with a given class of specialization. Each worker agent has its own class of specialization. The distance between agent’s and task’s specialization (compatibility) affects number of computational steps necessary to complete the task.

It is most efficient when agent has the same class of specialization as solved task. For example if agent is member of class $A$ and task being solved is from class $D$, the final compatibility $c$ can be shown as:

$$c_{A,D} = \text{diff}(A, D)$$

$$c_{A,D} = 1 + |\text{intval}(A) - \text{intval}(D)|$$

Where intval is function which converts a class type to an integer representation as shown in table 1.
Table 1: The conversion of class type to integer is used in diff function which measures agent–task compatibility. This table shows integer value for each class.

<table>
<thead>
<tr>
<th>Class of specialization</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer representation</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Each task computation can be interrupted when a new task request is accepted. In this case, the new task is enqueued by receiving worker and currently solved task is thrown away. It doesn’t guarantee that the accepted task will be computed as the first. Buffer is FIFO, tasks are taken from the head of queue. Next the manager is informed that given task should be offered once more to any worker. All workers have the same probability to be selected as task receiver.

The real-world tasks, such as some data mining or classifying problem have no explicitly defined class. It is hard to tell which method encapsulated in computing agent (eg. Neural Network, Decision Tree) is the best for current task. It will lead to requirement of exploration of task’s and agent’s metadata for computing agent-task compatibility.

3 Agent control

The worker agent has three basic parallel behaviours. The first one handles communication with manager, next one runs when agent has no active computation and takes the first task from agent’s buffer (if there is any) and starts new computation. The last one is computing loop core which makes one step from current task.

3.1 Communication and decision making

Communication part of behaviour is invoked when a message is delivered to worker agent. The agent has to make decision whether a task will be accepted. Based on this, agent sends proper answer message to the manager.

The decision is made as evaluation of an expression which connects weighted attributes of worker agent. The expression is represented by a tree. The main attributes in the expression are number of tasks solved by agent, computed percentage of actual task, agent-task compatibility of actual task, expected time to finish actual task, a load (average expected solving time per task in agent’s task buffer) and "happy", which means an average compatibility of enqueued tasks in agent’s buffer. Except the first one (long) are all variables real numbers. Shown variables are substituted into the tree at every run of decision process (when new task arrives) and the tree is evaluated as one real value. The result of decision is a boolean variable – accept or reject the offered task. The task is accepted when expression result is greater or equal to zero.

3.2 Computing loop

The computing loop is second main part of worker’s behavior. In each iteration some part of task is solved and appropriate agent’s attributes are actualized. As stated before, the loop can be interrupted and task can be thrown out. It is desirable to discover possible agent-task incompatibility as soon as possible. Completion of an extremely incompatible task costs more then stopping it and disposing it to a more compatible worker. But on the other hand, the interruption of current task is not good deal when worker agent has no other task in buffer.

This behaviour is planned by internal mechanism of JADE framework. Next run is scheduled by simple command at the end of each iteration of solving loop.

4 Genetic evolving of expression trees

The trees which represent decision expression are evolved by common genetic algorithm [2] for 300 generations. We used tournament selection. Size of population has only about 30–50 individuals because calculation of the fitness function is extremely expensive. It is necessary to reset all agents in MAS enviroment, prepare new task set and computation must race through it.

The number of tasks is constant for whole experiment and it was set up to 100 tasks. It is extremely small amount and there is considerable impact of random element. That means, if we have "luck", then most of tasks are send to suitable worker agents and they solve all tasks in short time. In this case the value of fitness function is significantly higher then in the other case, when major part of task is solved by less compatible agent. But bigger task set would lead to enormous experiment time (several days).
4.1 Construction and recombination of expression trees

The population of decision trees is initialized randomly. Each individual tree has all mentioned agent’s variables in its leaves. Each inner node represents one of the functions: ADD (addition), SUB (subtraction) or MUL (multiplication).

Cross and mutation operators are applied directly to the object of tree structure – there is no encoding such as binary for example. Every leaf node is a real number. Therefore it is important to apply operators very carefully, especially crossover operator.

Classic crossover between two individuals is not suitable, because it may produce invalid ancestors. For example there could be two same variables in the new expression tree. And therefore a crossover operator is applied to only one tree and exchanges two subtrees. In consequence it is a kind of mutation.

The application of mutation operator is different for leaf and inner nodes and also has different probabilities in each case. The mutation of inner nodes changes randomly function (ADD, SUB, MUL) in given node and is significantly less frequent than mutation of leaf nodes. The leaf node mutation adjusts value in node by small random number from interval $\pm 10\%$ of node value.

4.2 Fitness function

The fitness function combines two main criteria of experiment. The first is an average task time $T_{\text{avg}}$ and consists of time spent in computing loop and time for which the task waited in buffer. The second one is total experiment time $T_{\text{exp}}$ which includes computation and waiting time for all tasks. The fitness function can be computed as:

$$f = \alpha \cdot \frac{R_{\text{exp}}}{T_{\text{exp}}} + \beta \cdot \frac{R_{\text{avg}}}{T_{\text{avg}}}$$

where $\alpha, \beta$ are coefficients with actual values $\alpha = 2, \beta = 1$. We put higher weight on total time. Next $R_{\text{exp}}$ is empiric average value of total experiment obtained by 100 times run experiment with disabled ”brain” – the task acceptation was random with probability 50%. In a similar way $R_{\text{avg}}$ is average time per one task given from the same experiment.

Both $R_{\text{exp}}$ and $T_{\text{exp}}$ depend on task count. But fitness evaluation counts with it. It follows that random solution with accept rate 50% has fitness very close to 3.

5 Experiments and Results

Experimental task set consists of 100 tasks with random uniformly distributed type class. There are 5 classes and each of 5 workers is member of exactly one class. The most compatible task is solved by an agent in approximately 4150 ms. Regardless to agent–task compatibility the average time per task is 12450 ms. In both cases waiting time (in agent’s queue) of task is included. Without waiting time it is about 950 ms for the most compatible task, 2850 ms for average compatible task.
Table 2: Average computation times

<table>
<thead>
<tr>
<th>decision method</th>
<th>time per task [ms]</th>
<th>time for task set [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept 2% of tasks</td>
<td>1 004</td>
<td>99 432</td>
</tr>
<tr>
<td>accept 10% of tasks</td>
<td>8 593</td>
<td>882 961</td>
</tr>
<tr>
<td>accept 50% of tasks</td>
<td>12 450</td>
<td>1 458 572</td>
</tr>
<tr>
<td>accept 90% of tasks</td>
<td>16 299</td>
<td>1 648 996</td>
</tr>
<tr>
<td>accept 100% of tasks</td>
<td>16 453</td>
<td>1 691 372</td>
</tr>
<tr>
<td>best expression</td>
<td>1 013</td>
<td>120 636</td>
</tr>
</tbody>
</table>

For comparison of decisions based on evolved expressions we run the same task set on MAS, where worker agents make decisions randomly. They accept offered tasks with some level of probability.

Main objective is average elapsed time per one task (waiting in queue included) and total consumed time to solve the whole task set. As we can see from table 2, the total time depends approximately linearly on average task time.

Best evolved expression tree (shown at figure 1) has fitness 36,47158 and leads to average time per task of approximately 1013 ms. That means a very short waiting time of tasks. It is about 12 times better than random 50% accept solution. This perfect time is also caused by good agent–task compatibility during experiment. Simply each task was sent to suitable agent to solve. With many times repeated measurement the time grows to values around 3000 ms.

6 Conclusion

The shortest times are achieved when worker agent’s queues are very short. It leads to minimization of waiting part of task solving time. The best evaluated expression tree after 300 generations has approximately the same quality as situation when 98% of the tasks are rejected by worker agent in order to achieve the shortest possible queue.

In the future, we want to try to modify worker agents to solve some real tasks. There are many options which variables are suitable to be included in decision expression. Some variables could lead to better decision making.

Acknowledgement: I would like to thank to Mgr. Roman Neruda, CSc for initial idea and Mgr. Michaela Poková for language corrections. This work was supported by Grant No. SGS11/083/OHK3/1T/18 from the CTU in Prague.

References