Guidelines to integrate a DCOP algorithm for RMAS Benchmark

Fabio Maffioletti, Riccardo Reffato

Contents

1 Interface description

RMAS Benchmark is a tool written in Java that provides an easy way for the developer of DCOP algorithms to implement and test his own algorithm within the Robocup Rescue Simulator. In this first section we give a brief description of the interface and its components:

- CenterAgent: it represents a fictional agent, which is connected to the kernel and can send and receive messages like a normal agent. Unlike a real agent though, it does not compute its own assignment, but it functions as a "message distributor" between agents, providing a way for the agents to exchange messages without passing through the kernel (which has a limit of one message per cycle for each agent). It also sends the final message (the assignment for every agent) to the kernel.
- AssignmentSolver: the AssignmentSolver class is created by the CenterAgent, it has vision of the whole world, it creates the communication channel, the simulator of the decentralized problem and some utility

structures, in addition to keeping track of the statistics for the algorithm. After the end of the simulation it returns the assignment to the CenterAgent.

- AssignmentInterface: it is a general interface which provides a way to create both centralized and decentralized problem simulators. At this time only the decentralized one has been implemented.
- DecentralizedAssignmentSimulator: it implements the AssignmentInterface. It creates a number (specified in the gml map file) of agents, each one of them running the same DCOP algorithm and runs the simulation of the decentralized problem, calling the inizialize, send, receive and improveAssignment methods of each agent for a specified number of iterations (all of the iterations need to be completed by the end of a timestep of the kernel). In particuar the DecentralizedAssignmentSimulator first calls the initialize method of each agent, then it has a while loop in which it calls in this order: the send method of the ComSimulator for each agent, the receive method of the ComSimulator for each agent and the improveAssignment method for each agent. After the while cycle ends, it sets the selected target for each agent.
- ComSimulator: it provides a way for the DCOP agents to communicate without passing through the kernel.
- DecentralAssignment: it is an interface which provides the methods that the programmer's algorithm needs to implement in order to work in the rmas benchmark framework.
- AbstractMessage: it is a message interface, which the messages exchanged by the agents need to implement in order to work in the rmas benchmark framework.
- AssignmentMessage: it is a type of message the agents can use to communicate with each other.
- UtilityMatrix: this utility class represents a matrix that contains the local utility between each target and agent in the world model and provides several useful methods to retrieve agents and targets in the world. The utility is a simple estimation based on the fieryness of the fire (the less the fire is powerful the more the utility increases) and the distance between the agent and the fire.
- Stats: this class writes statistics of the algorithm to a file that can be found in the boot/logs directory of the benchmark. The tracked stats

are (for each step): number of burning buildings, number of buildings burnt at least once, number of destroyed buildings, destriyed area, number of violated constraints, CPU usage (number of millisecond to compute the assignment), exchanged messages in bytes, average NCCC (non-concurrent constraint checks).

2 How to integrate a DCOP algorithm with RMAS Benchmark

In this section we present a guide to integrate you algorithm with RMAS Benchmark.

1. Create a Java class file which implements the DecentralAssignment interface which is located in the RSLBench.Assignment package.
2. This class represents a proxy between your implementation of the DCOP algorithm and the RMAS Benchmark (you can put a jar file with the libraries you are using in the jars folder)
3. The class needs to implement the methods of the DecentralAssignment interface that we will briefly describe:
   - `inizialize(EntityID agentID, UtilityMatrix utilityM)`: in this method the agents are initialized and configured. It is called before the while loop, so the agents will be recreated at each kernel step. This method does not return.
   - `sendMessages()`: this method is called by the ComSimulator one time for each agent for each iteration of the while loop. In this method the agent computes and returns to the ComSimulator a Collection of the messages (AbstractMessage) it needs to send to the other agents to perform the assignment.
   - `receiveMessages(Collection<AbstractMessage> messages)`: this method is also called by the ComSimulator one time for each agent for each iteration of the while loop and the parameter id a Collection of the messages the agent needs to receive.
   - `improveAssignment()`: this method computes the assignment and returns a boolean set to true if the assignment is better than the last one.
   - `resetStructures`: the developer needs to implement this method if your algorithm has some structure you need to reset before the next initialization.

- `getAgentID()`: it returns the EntityID of the agent.
- tt getTargetID: it returns the EntityID of the last assigned target
- `getCCC`: this method computes the non concurrent constraint checks for each cycle of the algorithm. It returns an integer.

4. In the DCOP_algorithm.cfg file you can specify some parameters of the Benchmark execution such as: the base_package, the assignment_group and the assignment_class (these three parameters together will be built into a path that, from the RSL2/src directory will fetch the class in which you implemented you algorithm), the number_of_runs of the algorithm, the number_of_iterations of the algorithm for each step of the kernel, the experiment_start_time (the cycle in which the agents will start their computation) and other parameters (for more details see the DCOP_algorithm.cfg file). //to explain better after we have done it

5. Start your algorithm by running the run_DCOP.sh script

6. At the end of the run(s) you can find graphics of the tracked stats of the algorithm (possibly compared with DSA and MaxSum). //where?

3 Tracked statistics

The benchmark keeps track of some statistics of your algorithm and confronts them (possibly) with two states of the art algorithm: DSA and MaxSum. Here is an example of the statistics of the DSA algorithm with the simulation starting at time 50.

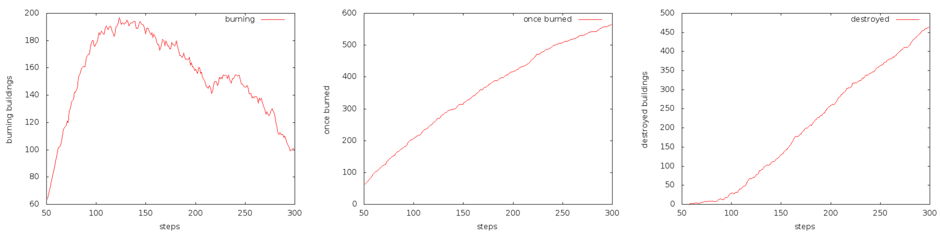

Figure 1: Respectively burning buildings, buildings which burned at least once and destroyed buildings

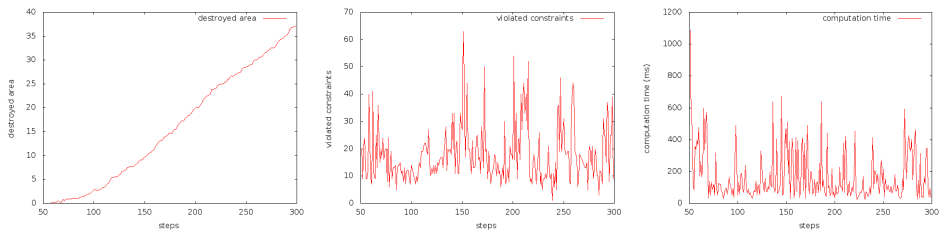

Figure 2: Respectively total destroyed area, the constraints violated and the computation time for each timestep

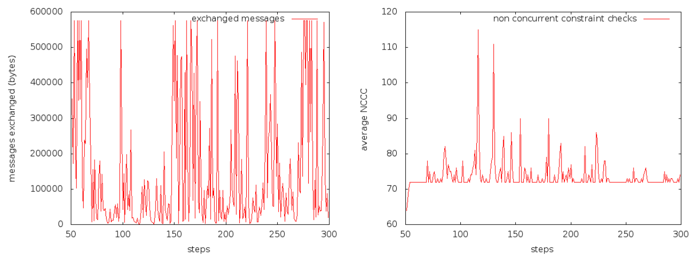

Figure 3: Respectively the messages exchanged between the agents for each timestep and the average NCCC

4 Future work

These are the weaknesses of the benchmark we found while working on it:

- ComSimulator: the ComSimulator needs to be re-implemented because it was thought to be used just for DSA. At this moment the ComSimulator builds autonomously clusters of agents which will communicate based only on range. Other algorithms, like MaxSum use other criteria to build these clusters. Moreover, at this moment an agent just needs to give the message to the ComSimulator and the ComSimulator knows which agents should read that message. It should be the agent instead to know who his neighbours are and to write the receivers of the message in the message itself, reducing the role of the ComSimulator to the one of a mailman, which reads the receivers on the message and delivers them.

- Agents: at this time the DecentralizedAssignment creates one instance of the DCOP algorithm for each PlatoonFireAgent in the map, but this can be restrictive for some algorithms, for example MaxSum, in which the PlatoonFireAgents are defined by the variables and not the DCOP instances (e.g. there can be one instance of MaxSum controlling every agent in the map).