ResQ Freiburg: Team Description and Evaluation

Alexander Kleiner Michael Brenner Tobias Bräuer
Christian Dornhege Moritz Göbelbecker Mathias Luber
Johann Prediger Jörg Stückler

Institut für Informatik
Universität Freiburg
79110 Freiburg, Germany

www.informatik.uni-freiburg.de/~rescue/

Abstract

ResQ Freiburg is the world champion of the 2004 RoboCup competition in the Rescue simulation league. RoboCupRescue is a large-scale multi-agent simulation of urban disasters where, in order to save lives and minimize damage, rescue teams must effectively cooperate despite sensing and communication limitations. To accomplish this, ResQ Freiburg introduced new methods for hierarchical path planning, death-time prediction of civilians, coordination of multi-agent city exploration, as well as an any-time rescue sequence optimization based on genetic algorithms. To evaluate the usefulness of these techniques we performed an extensive evaluation of the log files of the best participating teams in the competition. Our analysis explains the reasons for our team's success, and thus could also provide an evaluation tool for future competitions.

1 Introduction

The RoboCupRescue simulation league is a part of the RoboCup competitions and aims at simulating large scale disasters and exploring new ways for the autonomous coordination of rescue teams [6]. These goals are socially highly significant and feature challenges unknown to other RoboCup leagues, like the coordination of heterogenous teams with more than 30 agents, the search for civilians by the exploration of a large scale environment, as well as the scheduling of time critical rescue missions. Moreover, challenges similar to those found in other RoboCup leagues are inherent to the domain: The simulated environment is highly dynamic and partially observable by a single agent. Agents have to plan and decide their actions asynchronously in real-time.

This paper presents the approach of the *ResQ Freiburg* team, the winner in the RoboCupRescue simulation league at RoboCup 2004. The contributions of this work are methods for hierarchical real-time path planning, the prediction of the life-time of civilians based on *CART* [1] regression and *ADABoost* [4], the efficient coordination of an *active* disaster space exploration based on senses, communication and reasoning, as well as an any-time rescue sequence optimization based on genetic algorithms.

We compare the performance of our team with the performance of other teams in terms of their capability of, extinguishing fires, freeing roads from debris, disaster space exploration and civilian rescue. The evaluation is carried out with information extracted from simulation log files that were gathered during the RoboCup competition 2004. Our results explain clearly the success of our team, but also confirm the approaches proposed in this paper.

The remainder of this paper is structured as follows. We present the general agent architecture in Section 2. The following three sections concern general abilities of all agents: Path planning is described in Section 3, communication in Section 4, and exploration in Section 5. Specific capabilities are described in Section 6 (civilian rescue) and Section 7 (extinguishing of fires). In Section 8 we present some of the tools for developing our agents. Finally, an extensive evaluation and analysis of the 2004 RoboCupRescue competition is given in Section 9.

2 Agent architecture

The basic task of every agent (platoons as well as centers) is to collect, store and evaluate information, to choose actions fitting best to the situation and finally to execute these actions.

For storing information the agent maintains a detailed world model. Once connected to the kernel the world model consists of the initial information about the buildings, nodes and roads. It will be updated in every cycle by the sensing process. The agent "perceives" changes inside his hear and see radius, i.e. they are automatically communicated by the kernel, additional updates result from communication by other agents. Active sensing for specific tasks like the exploration will be explained later.

In order to flexibly handle the different duties of agents all of their abilities are specified in terms of *tasks*. A Task consists of its preconditions (describing executability of a task), the process itself (including actions to perform and messages to send), and a priority value (which may be updated as a result of new information). For example, firebrigade agents are able to extinguish buildings. Consequently, there is an *ExtuingishTask* which describes moving to the firescene, calculation and prediction of fire evaluation, and coordination with the other platoons. However, there are also more basic tasks (like *Move*) which are created by higher level tasks.

Creation of a task is triggered by agents (or existing tasks) as a result of updates to the world model and the agent's reasoning. However, there is no fixed decision hierarchy or control flow, but the decision process results from the ranking of tasks collected in a priority queue. To prevent frequent changes of decisions the current task can only be overruled by a task of a markedly higher priority or by a command from the station or a leading agent (cf. Section 7). Once a task has been performed the next in queue will be activated immidiately. If it took no effect for more than 3 rounds it is distroyed so that the agent is not stuck for too long.

3 Path planning

Path planning is a basic capability for every rescue agent wanting to reach its selected target. However, ResQ Freiburg agents heavily rely on their path planning capabilities already during target selection. They determine travel durations for almost all possible targets in advance, which enables them to evaluate the trade-off between a target's utility and travel duration. This allows the agents to take future travel costs into account very early during target selection, however, it means that an agent may query the path planner hundreds or sometimes thousands of times during one cycle!

We achieved the necessary efficiency for such frequent path planner queries with the three techniques described in this section. Firstly and foremost, planning is done on a graph considerably smaller than the original city map which consists only of the combined road segments between crossings, so-called "longroads". Secondly, instead of heuristic search we use an efficient implementation of Dijkstra's algorithm [2] on the longroad graph.

Counterintuitive as that may seem, this turns out to be effective as in every cycle every agent issues a great number of path queries all starting at the same point (its current position) which the path planner can answer in constant time after having computed and cached Dijkstra's algorithm just once. A simple extension on top of the standard algorithm enables agents to plan to and from locations that are not crossings. Thirdly, we use caching and object pools to prevent repeated computations of the same paths and, more importantly, to prevent the Java virtual machine from creating new objects repeatedly. Since these are low-level methods specific to the Java language they are not covered in this paper.

The specific usage of the path planner that results from the cycle-based movements of agents in the Robocup Rescue domain is described at the end of this section.

3.1 Planning with longroads

There are many possible locations between which agents can move on a Robocup Rescue map: buildings, streets, and street nodes. Paths between these positions are sequences of road-node tuples, possibly starting or ending with buildings (for a more accurate description cf. [7]). The most obvious way to compute such plans (and the way implemented in the YabAPI library [9]) is to search the graph spanned by those connected roads, nodes, and buildings. Note that edges in the transition graph do not correspond to roads, but to the adjacency information from the GIS which is communicated to the agents at the beginning of the simulation. In the transition graph, roads are just another kind of vertice! Using the given representation, the transition graphusually consists of several thousand nodes and edges. A (slightly simplified) example is shown in Fig. 1.

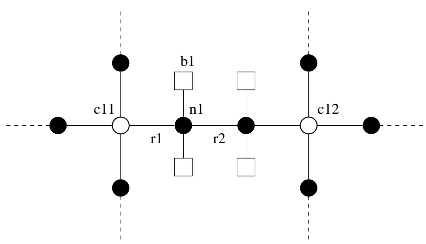

Figure 1: Roads, longroads, and crossings

In the non-hierarchical GIS representation that was chosen for RoboCupRescue, many nodes (e. g. $n1$) connect exactly two road segments. This is mainly due to the fact that buildings are represented as having an entry node in the Rescue simulation through which they are entered from the adjacent road or left again. We will call these nodes "internal" in the following.

The key realization underlying the ResQ Freiburg planner is that for an agent entering an internal node from one road segment there is no choice but to move to the second connected road segment (unless he wants to reach a building connected to the internal node). Intuitively, a set of road segments that are interlinked by internal nodes can be considered to be one single road, a "longroad". If agents enter a longroad and do not want to reach a building situated on the longroad they have no choice but to move on to the end of it. Longroads are terminated by crossings (more than two adjacent roads; for simplicity dead end nodes with only one adjacent road will be called crossings, too). In Fig. 1 internal nodes are shown in black and crossings in white. The route between crossings $c11$ and $c12$ constitutes a longroad which we will name $lr1$ in the following.

Since the only points of choice for the agent are crossings the ResQ planner builds a new transition graph consisting of crossings as vertices and longroads as edges. A longroad r

has two endpoints, its head and tail. An optimal path $P(c_1, c_2)$ between vertices on the new graph, i.e. crossings on the original one, can be found with any shortest-path algorithm and is easily recompiled to the original RoboCupRescue format by replacing longroads with the sequence of their internal nodes and roads, either from head to tail or from tail to head.

However, since the desired paths usually do neither start nor end at crossings some on-top processing is required. With the exception of buildings directly connected to crossings and crossings themselves, every location l on a map lies at exactly one longroad[1]. Consequently, the head and tail of this longroad, called c_l^1 and c_l^2 in the following, are the the only crossings directly reachable from l without passing another one.

Any route between two locations s and e (where here s and e shall be assumed not to be on the same longroad) must include either c_s^1 or c_s^2 and, respectively, either c_e^1 or c_e^2. The length of an optimal path from s to e therefore is:

$$\min_{i,j}\left(\overline{sc_s^i} + \overline{P(c_s^i, c_e^j)} + \overline{c_e^j e}\right)$$

To solve this formula efficiently, the ResQ planner precomputes and stores the direct routes from a location to its adjacent crossings. The optimal paths from the crossings c_s^1 and c_s^2 to *all* other crossings are computed and stored during two runs of Dijkstra's algorithm. As mentioned above this turns out to be quite efficient because during one cycle the agent usually queries many paths to locations all over the map. Single-target search could be speeded up by use of heuristics but would have to be repeated for every target. On the very sparse longroad graphs[2] twice running Dijkstra's algorithm turned out to be more efficient. Anyway, the longroad graphs computed from the maps currently used in the competition are so small (few hundred nodes and edges) that the the search process takes less time than the re-transformation of the path found to the kernel format.

3.2 Cycle-based planning

Efficiency is not the only criterium for path planning in the Robocup Rescue domain. Since the simulation is cycle-based, finding paths with minimal lengths or even minimal duration may not be the wisest choice. For example, two paths differing only by a few meters or, respectively, a few seconds can often be considered as equivalent as long as they take the same number of cycles to travel.

Therefore the ResQ path planner supports planning with cost functions that return real-numbers travel costs that can be interpreted as (fractions of) cycles. We have provided several such functions, accounting for or ignoring aspects like partially blocked roads, other agents, unknown road states, etc., that give good approximations of the travel time an agent has to expect according to its current knowledge of the world.

This allows the agents to build equivalence classes among paths and, consequently, targets. Several selection mechanisms allow to optimize other criteria when the numbers of expected cycles of the paths to a set of targets are equal or in some specific range. It is thus possible for an agent to select the most important target among the ones most easily reachable or, vice-versa, the closest among the most important targets.

4 Communication and world modeling

Communication fulfills two purposes. Primarily it is used for exchanging information about the current world state. Thus the agents obtain information that is observed by others and so improve their local world model. The second task of communication is the coordination

[1] In principle, buildings can have several entrances and that way may connect to several longroads, but to the best of our knowledge, not even the Rescue kernel currently does support this feature.

[2] The standard RoboCupRescue maps never feature more the four longroads meeting at a crossing.

of the agents. Agents can send instructions to other agents, in order to coordinate and optimize their acting.

In the RoboCupRescue simulation, platoon agents can only hear messages from platoons and the center of their own type. Furthermore, center agents can only hear messages from their platoons and other centers. Like in the real world the number of messages that can be sent or received in parallel is limited. A platoon agent can send or receive at most 4 messages in each cycle. The length of a message is limited to 256 bytes. A center agent can send or receive at most 2*n message in each cycle, where n is the number of platoon agents of the same type as the center.

In order to communicate as many different informations as possible in a single message and to allow easy dispatching of (parts of) its content to other agents, each message sent by a ResQ agent is composed of small self-contained units of information, called *tokens*. Each token consists of a header section and a data section. The header section stores type, priority, length and receivers of the token. The data section contains the information, which the token transfers. For instance there is a token of the type ROAD_BLOCKED, which transmits lists of blocked roads.

Both platoons and centers may initiate communication. Most commonly this is the case when the agent perceives changes in the world, i.e. receives updates to its world model from the kernel. For example, a platoon may initiate communication by composing a message of tokens and sending it to the respective station. The station, upon receiving the message, decomposes it to tokens again, interprets their content and, if necessary, updates its own local world model. After that the message is passed on to the other platoons of the same type and to other stations which may again route the information to their platoons.

In order to prevent the repeated passing on of a token, the stations store for each token whether it was already sent. Thus, although agents may receive the same information from different sources, they will only pass it on once, avoiding infinite cycles of communication.

Since platoons may accept only four messages per cycle it is impossible for platoons to rely on obtaining all information from their peers. Therefore most information is routed through the center agents. Direct communication between platoons takes place only in time-critical situations where instructions have to be passed quickly, i.e. between the firebrigades at a specific fire site.

5 Exploration

5.1 Knowledge Base

The Knowledge Base (KB) maintains the knowledge of an agent on the relation between the set of civilians C and the set of locations L. This is carried out by maintaining for each civilian $c \in C$ a set of locations L_c that contains all possible locations of the civilian. Furthermore, we maintain for each location $l \in L$ a set of civilians C_l that contains all civilians that are possibly situated at location l. Initially, $\forall c \in C, L_c = L$ and $\forall l \in L, C_l = C$.

The KB allows us the calculation of the expectation of the number of civilians situated at any location l. This is achieved by calculating the probability that civilian c is situated at location l, given the current state of the knowledge base:

$$P(loc(c) = l|KB_t) = \begin{cases} \frac{1}{|L_c|} & if l \in L_c \\ 0 & otherwise \end{cases} \tag{1}$$

Which yields the expectation on the number of civilians situated at location l:

$$E[|C_l|] = \sum_{i=0}^{|C|} P(loc(c_i) = l|KB_t) \tag{2}$$

Note that from the above follows that initially the expectation for each location l is given by $E[|C_l|] = \frac{|C|}{|L|}$, that is, we expect civilians to be uniformly and independently distributed on the map. This is clearly not the case if buildings have a different size or a different degree of their destruction. However, in order to incooperate this information, one would have to switch to a belief based, internal representation of the KB.

The KB is updated by either visual or auditory perception, communication of perception from other agents and reasoning. Both communicated and experienced perception are inserted into the KB with respect to the agent's sensor model. The sensor model returns for any location l the set of locations V_l and A_l that are in visual (10m) or auditory (30m) range of l, respectively. Based on the sensor model, one can perform either *positive* or *negative* update operations on the KB:

1. Positive updates:
   (a) Civilian c seen at l: $c = l \wedge c \notin L \setminus l$
   (b) Civilian c heard at l: $c \in A_l \wedge c \notin L \setminus A_l$
2. Negative updates:
   (a) No Civilian seen at l: $\forall c \in C \Rightarrow c \neq l$
   (b) No Civilian heard at l: *Not implemented*

The KB is implemented as a $|C|x|L|$ boolean matrix, where C is the set of civilians and L the set of locations. An entry $\langle c, l\rangle$ is set to *false* if a civilian c is definitely not at location l, and set to *true* otherwise (including the case of uncertainty). Initially, all entries are set to *true*.

5.2 District exploration

District exploration is a multi-agent behavior for the coordinated search of buried civilians. The behavior guarantees that at any time each agent is assigned to a reachable and unexplored district on the map. The search is carried out by all agent team (ambulance team, police force and fire brigade), if they do not have to perform tasks with higher priority, such as extinguishing or rescuing.

Coordinated exploration is difficult due to the fact that the blockage of roads and hence the reachability of regions is unknown to the agents in advance. To solve this problem, we implemented various clustering techniques, such as *agglomerative* clustering and *KD-tree* based clustering. These methods calculate from a given connectivity graph $G = \langle V, E\rangle$ of a city, where V represents the set of locations and E the set of connections between them, a hierarchical clustering. The hierarchical clustering, represented by a binary tree, provides at each level a partitioning of the city into n districts, reflecting the reachability of locations on the map. Note that the clustering has to be revised continuously, because the state of the roads and thus the connectivity changes during each cycle of the simulation.

Since agent teams are assigned arbitrarily to the exploration task[3], the total number of assigned agents is usually unknown. Therefore each team performs a separated clustering of the map that overlaps with the clusterings of other teams. From the clustering, agents are uniquely assigned to a district within a team, if there are less districts than agents, districts are assigned multiply. In order to prevent agents from exploring locations twice, it is necessary to further coordinate the search by the negotiation of exploration targets (see section 5.5).

The exploration within a district can significantly be accelerated if exploration targets are selected with respects to the agent's sensor model. The sensor model provides for each location on the map the set of locations that can be observed from that location. In order

[3] Agents might be busy with more important tasks, such as extinguishing fires

to foster the selection locations with high entropy, a utility value $U(l)$ is calculated that is equal to the number of locations $|O_l|$ observable from location l:

$$U(l) = |O_l| \tag{3}$$

The overall sum of utilities over time can be maximized by the selection of targets with high utility as well as targets that are reachable within a short amount of time. Hence, from the set of locations L_D that are within the agent's district, a target location l_t is decided based on the trade-off between utility $U(l)$ and travel cost $C(l)$:

$$l_t = \underset{l \in L_D}{argmax}\, U(l) - \alpha * C(l) \tag{4}$$

whereas α is a constant regulating the trade-off between the estimated travel costs and the exploration utility and has to be determined experimentally. The estimated Travel costs $C(l)$ are provided by the path planner, described in section 3.

5.3 Active Exploration

Active exploration is an extension to the previously described district exploration task in that the search focuses on locations with high evidence on civilian whereabouts. This is carried out by exploiting the knowledge, collected from senses, communication and reasoning, in the KB (see section 5.1). Evidence from the KB is utilized by calculating the utility value $U(l)$ in equation 4 with respect to the number of civilians expected to be found at all observable locations O_l:

$$U(l) = \sum_{k \in O_l} E[|C_k|] \tag{5}$$

which yields, after inserting equation 2:

$$U(l) = \sum_{k \in O_l} \sum_{i=0}^{|C|} P(loc(c_i) = k | KB_t) \tag{6}$$

5.4 Active surveillance

Furthermore it is important for the rescue team to have up-to-date information on the injured civilians that have been found by the exploration task. The prediction module, described in section 6.1, can provide as accurate predictions of the civilian life time, as up-to-date the information on *buriedness*, *damage* and *hitpoints* is. As we will describe in section 6.2, the number of civilians that can be rescued depends on the efficiency of the rescue team, which in turn, depends on the accuracy of predictions. Hence we extended the active exploration behavior in that it assigns agents to the survilliance of known civilian locations after the map has been explored sufficiently. The survilliance behavior is carried out by sampling interesting locations randomly from the set of known civilian locations, whereas locations with obsolete information are selected with high probability.

The number of agents that are assigned to active search is limited to $\frac{|L|}{k}$, where L is the set of open locations and k a constant that has to be determined experimentally. A small k might cause the agents to waste resources on the search, whereas a large k increases the time needed to finish the search. All agents above the assignment limit are performing active surveillance.

5.5 Team coordination

Besides the agent distribution due to the assignment of districts, it is necessary to further coordinate the multi-agent search in oder to prevent the multiple exploration of locations.

This is carried out by communicating the information on found civilians, as well as locations that have been visited. However, if agents are selecting exploration targets from the same district (i.e. due to the overlap or the shortage of available targets), it might still occur that they explore locations twice. We implemented two methods for reducing the probability of multiple target exploration. Firstly, agents select exploration targets from a probability distribution. Secondly, agents negotiate targets they plan to explore in the next cycle via the short range communication channel (*say* and *hear*).

It turned out that the latter performs poor, if agents are able to move much longer distances in a cycle as they are able to observe, which is true for the current parameter setting of the RoboCupRescue kernel. The problem could be solved by performing the negotiation via the long range communication. Unfortunately, this does no pay off, since communication is a limited resource. Hence agents decide their exploration targets by a random selection that prefers targets that yield high score after equation 4.

6 Civilian Rescue

6.1 Lifetime prediction

To achieve good results in the civilian rescue process, it is necessary to know when a civilian will die. If there is a reliable prediction for the life time of a certain civilian the scheduling of the rescue operation can be adapted accordingly. On the one hand, it is possible that a civilian does not need to be rescued at all, because it is alive at the end of the simulation. On the other hand, it is possible that a civilian will die within a short time and has to be rescued as soon as possible in order to survive.

For the ResQ Freiburg agents, machine learning was used to gain a prediction for the civilian's life time and classification into survivors and victims. We created an *autorun tool* that starts the kernel and the agents simultaneously in order to collect arbitrary data. The tool was used for several simulation runs on the Kobe, VC and Foligno maps, from which a large amount of datasets were generated. A data set consists of the values for *health* and *damage* of each civilian at each time step gained during the simulation. In order to reduce the noise in the data, simulations were carried out under the following conditions:

- Simulating time: 400
- No rescue processes by the agents
- No fires

The latter two are necessary in order to prevent unexpected changes of the damage of a civilian due to its rescue, resulting in zero damage, or due to fires, resulting in unpredictable high damage. For the calculation of the life time, there has to be determined a time of death for each dataset. Hence, the simulation time was chosen to be 400 rounds which seemed to be a good compromise between an ideal simulation time of ∞ and the standard simulation time of 300 rounds that would lead to a non-uniform distribution of the datasets.

Regression and classification was carried out with the *WEKA* [10] machine learning tool. We utilized the *C4.5* algorithm (decision trees) for the classification task. The regression of the simulation time is based on Adaptive Boosting (Ada Boost) [4]. Since the current implementation of the *WEKA* tool does only provide Ada Boost on classification, we had to extend this implementation for regression [3], which then has been applied with regression trees (CART) [1].

The regression trees have been evaluated on test data sets in order to learn the confidence of a prediction in dependency of the civilian's damage and the distance between the timestamp of the dataset and the predicted time of death. Confidence values are necessary, since predictions are less accurate as higher the difference between the observation and the civilian's actual time of death. The sequence optimization, described in section 6.2, relies on the confidence values in order to minimize sequence fluctuations.

6.2 Genetic Sequence Optimization

If the time needed for rescuing civilians and the life time of civilians is predictable, one can estimate the overall number of survivors after executing a rescue sequence by a simulation. For each rescue sequence $S = \langle t_1, t_2, ..., t_n \rangle$ of n rescue targets, an utility $U(S)$ is calculated that is equal to the number of civilians that are expected to survive. Unfortunately an exhaustive search over all $n!$ possible rescue sequences is intractable. A straight forward solution to the problem is, for example, to sort the list of targets by the time necessary to reach and rescue them and to subsequently rescue targets from the top of the list. However, as shown in section 9, this might lead to sub-optimal solutions. Hence we decided to utilize a Genetic Algorithm (GA) for the optimization of sequences and thus the subsequent improvement of existing solutions [5].

The time for rescuing civilians is approximated by a linear regression based on the buriedness of a civilian and the number of ambulance teams participating to the rescue. Travel costs between two targets are estimated by averaging over costs sampled during previous simulation runs. This is much more efficient than the calculation of exact travel costs, involving in the worst case the calculation of the Floyd-Warshall matrix[4].

The GA is initialized with heuristic solutions, as for example solutions that *greedily* prefer targets that can be rescued within a short time or urgent targets that have a short lifetime. The fitness function of solutions is set equal to the previously described utility $U(S)$. In order to guarantee that solutions in the genetic pool are at least as good as the heuristic solutions, the so called *elitism* mechanism, which forces the permanent existence of the best solution in the pool, has been used. Furthermore we utilized a simple one-point-crossover strategy, a uniform mutation probability of $p \approx 1/n$ and a population size of 10. Within each cycle, 500 populations of solutions are calculated by the ambulance station from which the best sequence is broadcasted to the ambulance teams that synchronously start to rescue the first civilian in the sequence.

One difficulty of the sequence optimization is given by the fact that information stored in the KB on civilians changes dynamically during each round and thus might cause fluctuations of the rescue sequence. This can be caused by two reasons: Firstly, civilians are discovered by the active exploration, which is executed by other agents at the same time. Secondly, predictions vary due to information updates from active or passive survilliance. The latter effect can be weakened by updating the sequence with respect to the confidence of predictions. Updates of the information on civilians are ignored, if they are not statistically significant with respect to their confidence interval.

The effect of information updates due to exploration has to be controlled by deciding between rescue latency and rescue permanence. Therefore we implemented a reactive mechanism that recognizes from information updates any emergency rescue targets and, if one has been detected, subsequently causes a revision of the current target sequence. An emergency situation is given if any other target has to be rescued immediately in order to survive, but also the current target would survive if postponing its rescue.

7 Extinguishing fires

7.1 Prediction

To fight fires effectively a good prediction of the fire spread is essential. But in order to make this prediction, we need two pieces of information for each building: a) the set of surrounding buildings and b) the time it takes for a fire to spread to those neighbors.

For the latter, we determined that the time until a building reaches fieriness 2 (which we dubbed *orange time*) is a useful quantity. It does not only allow us to compute the fire-start

[4]The Floyd-Warshall matrix calculates in $O\left(n^3\right)$ from all sources to all targets the shortest path.

time but it is also a good indicator of the difficulty of the building. To compute the orange time, we used automated data-mining techniques as outlined in section 5.5.

For the problem of computed the fire neighborhood, we use a algorithm which checks the distances between the walls of the source building and another one. If the distance is smaller than a value d, it checks if other buildings are in between. If this is not the case, the second buildings is added to the *fire neighbourhood* of the source buildings. To improve the speed, only buildings whose center is inside a radius r from the source are considered. But as this calculation is only performed once at the connect time, performance is not a high priority.

To predict the fire spread, a modified *Dijkstra's algorithm*, where all burning buildings are put into the queue at the start, is used. The distance $d(A, B)$ between two Buildings A and B is defined as the time is takes for building A to put building B on fire. If fire cannnot jump over from A to B, $d(A, B)$ is infinite.

Additionally, the algorithm is used to implement a clustering on the fires. Using *union find structures* two fire sites that will touch in N turns are merged. So, at the beginning of the simulation every fire will be in its own site, and merging will take place, as the fire spreads.

7.2 Target selection

Target selection is implemented in two stages: first, a fire site is selected, then a specific building in that site.

Site selection is based on the cost/utility-ratio, where C is the sum of the orange times of all buildings in the fire site and $U = b + \alpha * civs$ where b is the number of buildings that will be put on fire by the site in the next turns, and $civs$ is the number of civilians in these buildings.

Additionally, buildings are added to the costs that are predicted to catch fire before the agent can reach the fire site.

In the second stage, a fire on the selected site is chosen. To do this, we first remove all fires from the site that have no unburned fire neighbourhood or have $fieryness = 3$. Fires that cannot be extinguished with the available agents are dropped as well as fires that are not reachable. The remaining fires are then ordered by a priority function

$$P(b) = (\alpha * unburned + \beta * extinguished + \gamma * civs + f(dist)) * g(direction)$$

where $unburned$ and $extinguished$ are the respective number of buildings in the fire neighbourhood, $civs$ the number of living civilians in an unburned building and f is a function that computes a bonus or malus depending on the distance between the agent and the fire. g is a function that penalises fires that will spread towards the edge of the map, with g approaching zero the nearer the building is to the edge.

7.3 Coordination

Due to world-model differences and different starting positions, agents might select different buildings to extinguish. As this is usually unwanted, we introduced several mechanisms to coordinate the fire brigades.

Every time a fire brigade agent switches to a different fire site (or chooses one for the first time) it sends a message to the station containing the site. The station will collect these messages and send a summary of the number of brigades at every site to the agents every few turns. The agents will then try to switch to a reachable site with the largest number of other fire brigades.

In addition to this, each agent tells the station for every site if it is reachable or not. If an agent is refuelling, no site is reachable. From this messages the station computes the maximum number of available agents for each site and sends it to the fire brigades every

Figure 2: The ResQ-Viewer with several plug-ins

turn. The brigades can then use this information to avoid extinguishing buildings they cannot put out with the available number of agents.

We also implemented a leader mechanism. Each fire site – if there are any agents in it – has a platoon leader who sends his current and next target to the other agents. As these orders are usually very time-critical, they are not relayed over the fire station. The other agents will then try to extinguish the same fire as their leader. If this is not possible for some reason, or the building is already put out, they will fall back to the next buildings the leader sent – more fall-backs are usually not necessary, as the message latency is normally only one turn.

The leaders are assigned by the station using the following scheme:

- If a site has no leader, the first agent who switches to this site becomes leader.
- If a leader has to refuel, he resigns leadership two turns beforehand but continues to send orders in this time. The other agents that are not in the refuge then send their remaining water quantity to the station, which then announces the agent with the most water as the new leader.

8 Development and debugging tools

8.1 Viewer tools

In order to be able to program and debug a system as complex as a RoboCup Rescue agent system, we developed an extensive tool-set based on the viewer by T.Morimoto [8].

Every module in our agents is able to write status and debug data to a number of log-files in either human- or machine-readable form. Additionally every agent writes each change in his world-model – updates from the kernel, from communication with other agents and from its own reasoning – to a world log.

We heavily modified the Moriomoto-Viewer, so our viewer is able to maintain several world-models and several viewer windows at the same time. This enables us, using the world log from the agents, to display the internal world model of each agent alongside the real kernel view.

Another important development was the design of a plug-in API, that allowed us to extend the capabilities of the viewer anytime without getting problems with maintainability. Every plug-in is attached to a viewer window and has access to its world model. It can also interact with the viewer by receiving mouse clicks from the window and mark objects in different colours and styles. By reading the log-files, a plug-in can then show additional information about the internal state of an agent, either in the viewer window (e.g. its current target or the planned path) or in its own window (e.g. the task-queue). Furthermore, the plug-in-system allows us to implement or modify some of our algorithms in the viewer by writing an appropriate plug-in. This makes development much easier, as you don't have to start your agents and evaluate their performance after every change, and you can get a direct feedback to parameter changes without having to change the source-code and recompile.

8.2 Batchcontroller

The batchController software[5] is designed for data mining in the RobocupRescue domain. The module starts simulation runs automatically and logs formatted data specific to a given learning task.

A task receives pre- and postcondition calls in each cycle to evaluate task-specific conditions on the world and thus determine the objects that are to be logged or that the logging is to be stopped for. Each task has a name that is used to produce unique log file names automatically.

9 Results

During the competition, teams are evaluated by an overall score that is calculated based on the state of civilian health and building destruction. However, since this score incooperates the total performance of all agent skills, such as exploration, extinguishing and rescuing, it is difficult to assess single agent skills directly. In order to compare our agents with agents from other teams, the performance of typical agent skills are emphasized by an evaluation of log files that were collected during the 2004 competition. The following tables provide results from all rounds of all teams that passed the preliminaries. All values are concerning the last round, i.e. the percentage of clean roads at round 300. Bold numbers denote the best results that have been achieved during the respective round.

Table 1 shows the percentage of blockades that have been removed by the police agents. The results show that particularly the teams *Damas Rescue* and *The Black Sheep* most efficiently removed blockage from the roads.

Table 2 shows the percentage of buildings that have been saved by the fire brigades. Obviously the team *Damas Rescue* saved most of the buildings, whereas *SBC* reached a robust behavior, shown by the good average value.

The efficiency of exploration is another important criterium for the team evaluation. As more locations of civilians are known, as more efficiently rescue operations can be scheduled. Table 3 shows the percentage of buildings that were visited by agents[6].

The result shows that *Caspian* explored most of the buildings. However, the percentage of explored buildings does not necessarily correlate with the percentage of found civilians,

[5] Full documentation at http://kaspar.informatik.uni-freiburg.de/ project2/batchController/batchController.html

[6] Note that full communication of visited locations as well as exploitation of a sensor model was assumed

Table 1: Percentage of clean roads

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	74,68	**82,22**	71,79	70,43	N/A	N/A	N/A	N/A
Final-Random	77,84	**86,51**	77,66	63,10	N/A	N/A	N/A	N/A
Final-Kobe	92,25	**93,74**	92,08	92,05	N/A	N/A	N/A	N/A
Final-Foligno	96,41	**97,72**	97,22	96,07	N/A	N/A	N/A	N/A
Semi-VC	67,93	**79,57**	68,86	57,90	67,22	57,85	53,27	80,53
Semi-Random	82,53	**87,44**	77,47	81,93	82,26	79,53	80,30	78,76
Semi-Kobe	92,40	93,65	92,71	92,51	92,62	92,56	93,55	**99,72**
Semi-Foligno	95,45	**97,08**	95,58	96,37	96,93	97,07	95,92	83,44
Round2-Kobe	92,52	93,52	91,46	92,46	92,78	93,45	92,25	**99,50**
Round2-Random	87,74	90,03	87,62	87,71	87,86	88,73	85,03	**99,97**
Round2-VC	91,34	91,62	90,74	89,87	91,40	90,92	N/A	**98,86**
Round1-Kobe	89,19	89,51	87,78	88,21	88,30	87,70	**91,12**	81,17
Round1-VC	91,90	92,13	91,74	91,84	N/A	91,81	91,54	**99,82**
Round1-Foligno	95,84	96,92	96,52	96,36	94,19	96,62	**97,63**	80,15
Number of wins	0	**7**	0	0	0	0	2	5
AVG %:	87,72	**90,83**	87,09	85,49	88,17	87,62	86,73	90,19
STD %:	8,25	**5,09**	8,59	11,25	8,93	11,59	13,63	9,96

Table 2: Percentage of saved buildings

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	47,21	54,13	**81,67**	43,19	N/A	N/A	N/A	N/A
Final-Random	24,04	**26,38**	15,03	12,35	N/A	N/A	N/A	N/A
Final-Kobe	38,24	**61,89**	38,38	13,51	N/A	N/A	N/A	N/A
Final-Foligno	**91,15**	62,77	60,92	34,56	N/A	N/A	N/A	N/A
Semi-VC	23,45	23,60	25,49	27,14	19,12	25,10	26,36	**27,22**
Semi-Random	23,18	**28,73**	18,09	19,55	22,82	21,45	17,09	18,91
Semi-Kobe	**96,49**	76,76	94,32	95,41	24,32	90,54	55,27	94,19
Semi-Foligno	36,22	**38,06**	32,72	37,79	31,89	28,48	26,82	23,23
Round2-Kobe	70,27	37,03	59,73	95,41	48,38	61,49	10,54	**95,54**
Round2-Random	99,04	60,91	54,68	99,16	63,55	97,60	80,70	**99,52**
Round2-VC	10,23	11,57	10,23	13,53	12,67	**71,99**	N/A	36,51
Round1-Kobe	99,46	98,92	**99,73**	**99,73**	99,05	98,78	67,16	91,89
Round1-VC	97,25	99,53	79,70	**99,76**	N/A	98,90	99,53	99,53
Round1-Foligno	**98,99**	**98,99**	36,13	45,99	32,53	54,29	43,59	29,86
Number of Wins:	3	**5**	2	2	0	1	0	3
AVG %:	61,09	55,66	50,49	52,65	39,37	**64,86**	47,45	61,64
STD %:	37,80	34,11	31,83	37,50	27,28	**31,63**	30,49	36,70

as shown by table 4[7]. This is due to the fact, that communication as well as reasoning might increase the efficiency of exploration. At the end, more civilians were found by *ResQ Freiburg* than *Caspian*, even the latter explored more buildings.

Important for efficient rescue operations, is the point in time when civilian whereabouts are known. As earlier civilians are found as better their rescue can be scheduled. Figures 3 and 4 show the number of civilians found during each cycle on the *RandomMap*. The results confirm the efficiency of *ResQ Freiburg's* exploration: At any time the agents knew about more civilians than agents of any other team.

Figure 5 documents the difference between a greedy rescue target selection, i.e. to prefer targets that can be rescued fast, and the selection based on an optimization by a genetic algorithm. It can be seen that an optimization of the rescue sequence clearly increases the number of rescued civilians.

Finally table 5 shows the number of civilians saved by each team: *ResQ Freiburg* saved more than 620 civilians during all rounds, which are 35 more than the second best and 59 more than the third best in the competition.

[7]Note that civilians are considered as being found, if one of the agents was within their visual range

Table 3: Percentage of explored buildings

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	83,48	83,24	**87,02**	67,27	N/A	N/A	N/A	N/A
Final-Random	69,62	72,62	**78,13**	49,92	N/A	N/A	N/A	N/A
Final-Kobe	89,19	92,97	89,73	**94,19**	N/A	N/A	N/A	N/A
Final-Foligno	84,15	85,25	**86,73**	74,29	N/A	N/A	N/A	N/A
Semi-VC	69,39	72,86	**77,42**	45,08	52,01	52,87	47,92	59,72
Semi-Random	**78,91**	68,73	71,91	54,36	59,36	70,27	46,18	46,18
Semi-Kobe	85,41	96,22	92,97	95,54	66,62	97,30	**99,46**	91,89
Semi-Foligno	74,75	89,12	84,98	62,49	65,35	**92,53**	79,08	20,74
Round2-Kobe	87,16	90,68	95,00	91,76	80,54	94,19	**99,46**	92,43
Round2-Random	81,18	80,94	88,61	84,53	60,67	**94,24**	82,61	87,89
Round2-VC	83,40	70,18	84,58	40,44	67,74	87,88	N/A	**89,54**
Round1-Kobe	87,43	90,27	94,05	96,08	96,62	97,70	**97,84**	80,95
Round1-VC	85,37	90,48	95,28	94,26	N/A	97,72	**100,00**	91,35
Round1-Foligno	83,78	**90,05**	**90,05**	60,00	54,65	88,57	67,37	13,00
Number of Wins:	1	1	**4**	1	0	2	4	1
AVG %:	81,66	83,83	**86,89**	72,16	67,06	87,33	79,99	67,37
STD %:	5,82	9,98	**7,87**	22,21	13,87	14,59	21,84	30,80

Table 4: Percentage of found civilians

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	97,22	94,44	**100,00**	81,94	N/A	N/A	N/A	N/A
Final-Random	**90,91**	85,71	81,82	70,13	N/A	N/A	N/A	N/A
Final-Kobe	**98,77**	97,53	95,06	**98,77**	N/A	N/A	N/A	N/A
Final-Foligno	**96,67**	**96,67**	**96,67**	72,22	N/A	N/A	N/A	N/A
Semi-VC	77,92	77,92	**85,71**	45,45	53,25	53,25	50,65	63,64
Semi-Random	**88,51**	73,56	72,41	63,22	67,82	80,46	52,87	55,17
Semi-Kobe	**100,00**	**100,00**	**100,00**	98,61	79,17	**100,00**	**100,00**	97,22
Semi-Foligno	90,12	95,06	86,42	81,48	83,95	**97,53**	85,19	30,86
Round2-Kobe	98,89	98,89	97,78	95,56	91,11	**100,00**	**100,00**	98,89
Round2-Random	**98,89**	95,56	**98,89**	81,11	70,00	96,67	85,56	94,44
Round2-VC	**92,22**	78,89	90,00	45,56	72,22	88,89	N/A	87,78
Round1-Kobe	94,29	**100,00**	**100,00**	98,57	**100,00**	**100,00**	94,29	78,57
Round1-VC	**100,00**	**100,00**	**100,00**	97,14	N/A	**100,00**	**100,00**	98,57
Round1-Foligno	**100,00**	97,14	94,29	77,14	74,29	92,86	77,14	14,29
Number of Wins:	**9**	4	7	1	1	5	3	0
AVG %:	**94,60**	92,24	92,79	79,06	76,87	90,97	82,85	71,94
STD %:	**7,17**	10,53	9,03	20,75	13,73	14,69	19,35	30,25

10 Conclusion

The results presented in Section 9 clearly show the strengths of the *ResQ Freiburg* team – as well as some points for future improvement. We are confident that an efficient exploration and rescue sequence optimization are crucial components of the overall team performance. It is interesting to note that during the final on the *RandomMap*, which decided by unbelievable 0.4 points of the total score the positioning between *Damas Rescue* and *ResQ Freiburg*, *ResQ Freiburg* was able to rescue even seven civilians more than the second best.

However, the basis of the *Knowledge Base* and thus the exploration is built upon the agents' world model and efficient information exchange by communication. In order to gain a reliable world model, one has to implement various tools in order to optimize and understand the information processing in the background. The implementation of communication and world modeling was probably one of the tasks we had to spend most of our time on.

Another problem that had to be tackled by our team was the strong limitation on computational resources. Without an efficient implementation of the path planner, our team, whose code is entirely written in JAVA, would not be able to compete.

In summary, the results presented provide an interesting insight in the 2004 competition: Besides strategies for extinguishing fires and the removal of blockades, also explo-

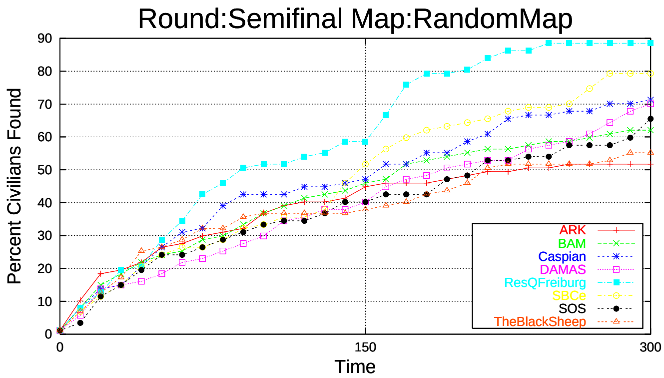

Figure 3: The number of civilians found by exploration on a randomly generated map during the semi-final

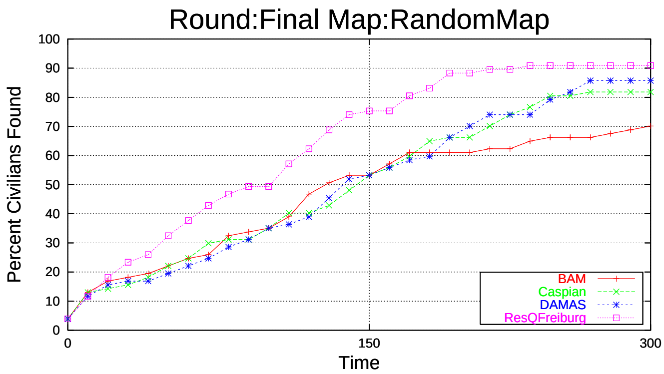

Figure 4: The number of civilians found by exploration on a randomly generated map during the final

ration and sequence optimization are crucial subproblems of the RoboCupRescue simulation league.

References

[1] L. Breiman, J.H. Friedman, R. A. Olshen, and C.J. Stone. *Classification and regression trees*. Wadsworth & Brooks, 1984.

[2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[3] Harris Drucker. Improving regressors using boosting techniques. In *Proc. 14th International Conference on Machine Learning*, pages 107–115. Morgan Kaufmann, 1997.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

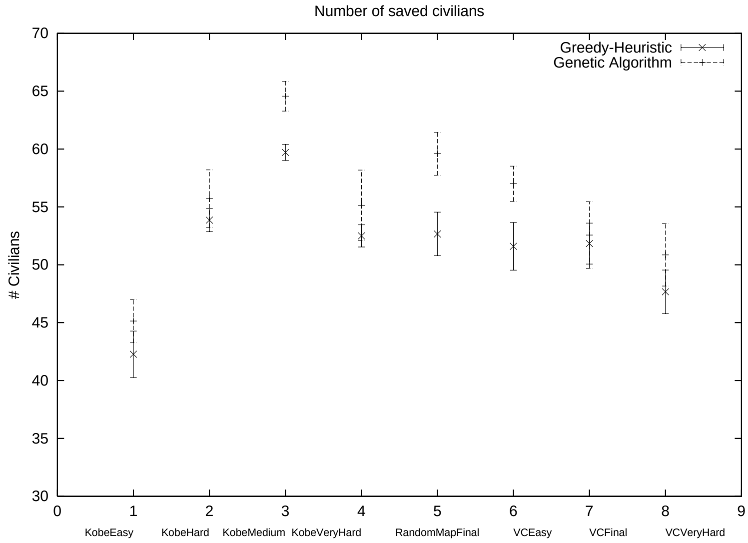

Figure 5: The number of rescued civilians under two different strategies

Table 5: Number of saved civilians

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	42	43	**52**	34	N/A	N/A	N/A	N/A
Final-Random	**32**	25	29	16	N/A	N/A	N/A	N/A
Final-Kobe	**46**	45	**46**	30	N/A	N/A	N/A	N/A
Final-Foligno	**66**	54	50	29	N/A	N/A	N/A	N/A
Semi-VC	**18**	15	17	12	11	12	12	14
Semi-Random	22	**26**	16	14	20	14	15	15
Semi-Kobe	**57**	47	54	52	20	39	34	44
Semi-Foligno	37	**46**	44	43	42	28	29	24
Round2-Kobe	**57**	37	43	50	43	35	28	43
Round2-Random	**52**	48	39	45	47	44	50	37
Round2-VC	31	33	32	24	37	**51**	N/A	34
Round1-Kobe	45	**51**	47	43	47	31	25	34
Round1-VC	**62**	**62**	55	57	N/A	51	54	44
Round1-Foligno	**53**	**53**	37	33	37	41	30	23
#Wins:	**9**	5	2	0	0	1	0	0
Σ TOTAL:	**620**	585	561	482	304	346	277	312
Σ SEMI+PREM	**434**	418	384	373	304	346	277	312

[5] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[6] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. RoboCup Rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE Conf. on Man, Systems, and Cybernetics(SMC-99)*, 1999.

[7] T. Morimoto. *How to Develop a RoboCupRescue Agent*, 2002. http://ne.cs.uec.ac.jp/~morimoto/rescue/manual/.

[8] T. Morimoto. *RoboCup Rescue Viewer*, 2002. http://ne.cs.uec.ac.jp/~morimoto/rescue/viewer/index.html.

[9] T. Morimoto. *YabAPI agent development kit*, 2002. http://ne.cs.uec.ac.jp/~morimoto/rescue/yabapi/.

[10] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.