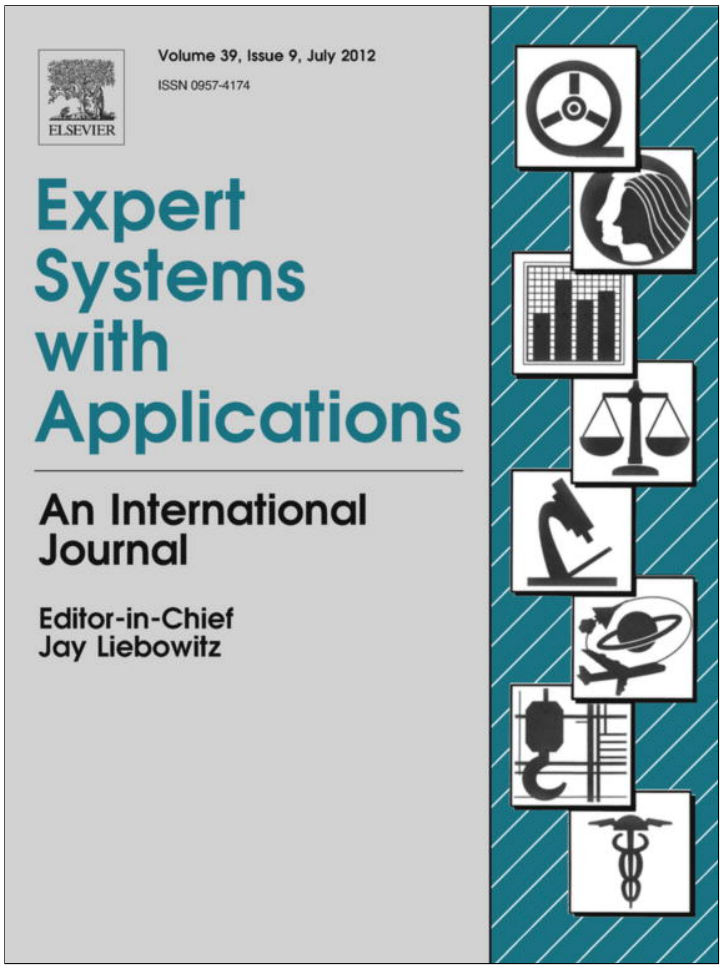

Expert Systems with Applications 39 (2012) 7887–7894

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

# Temporal bounded reasoning in a dynamic case based planning agent for industrial environments

Martí Navarro [a], Juan F. De Paz [b,*], Vicente Julián [a], Sara Rodríguez [b], Javier Bajo [b], Juan M. Corchado [b]

[a] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de la Vera s/n, Valencia 46002, Spain
[b] Departamento Informática y Automática Universidad de Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain

ARTICLE INFO

*Keywords:*
Multiagent systems
Temporal bounded reasoning
Case-based reasoning
Ambient intelligence

ABSTRACT

This paper presents a planning model integrated within a TB-CBP-BDI real-time intelligent agent that provides special abilities for planning in a predictable time, which makes its use especially appropriate in systems where certain temporal constraints must be satisfied. The proposed TB-CBP-BDI real-time agent is the core of a multi-agent system that manages security issues in industrial environments, where time constraints are a key factor. The proposed planning model facilitates the automatic temporal bounded reorganization of tasks to provide the system with adaptation abilities to the changes that occur in the environment. The planning mechanism focuses on optimizing industrial and manufacturing processes, specifically the tasks performed by the available security entities in these environments. Additionally, several Ambient Intelligence technologies such as QR-CODES, GPS, Wi-Fi and HSDPA are used to develop the intelligent environment that was tested and analyzed in this study.



## 1. Introduction

Over the last few years, the use of artificial intelligence techniques to control industrial processes has become increasingly relevant (Sáenz et al., 2008). However, there are some aspects that still need to be improved, especially those related to the efficiency of techniques and technologies used to monitor security activities. The implementation of time control systems has had a positive influence on productivity in industrial environments (Zheng, Wang, & Xue, 2009), since the workers optimize their potential and enhance the processes on which they collaborate. Remote monitoring is becoming increasingly common in industrial scenarios, where recent studies (Inology, 2005) reveal that at least 3% of the workday is lost, allowing supervisors to observe the behavior of remote workers and the status of the facilities. Multi-agent systems (MAS) (Partalas, Feneris, & Vlahavas, 2008; Wooldridge & Jennings, 1995; Zheng et al., 2009) have been recently explored as supervisor systems, with the flexibility to be implemented in a wide variety of devices and scenarios, including industrial and manufacturing environments. Moreover, the application of this paradigm seems to be especially appropriate for solving complex problems which require intelligent and temporal bounded responses, as is the case with some typical industrial or manufacturing scenarios (Stankovic, 1998; Zheng et al., 2009). In these kinds of scenarios, the execution of a task after its temporal deadline is completely useless or decreases the quality of the solution; consequently tasks should be designed as a real-time problem. MAS are explored in this article because they can incorporate flexibility and distribution in real-time problems. From a formal point of view, a real-time system (RTS) is a system in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced (Carrascosa, Bajo, Julian, Corchado, & Botti, 2008; Stankovic, 1998). A typical RTS is commonly known to be made up of a set of tasks characterized by a deadline, a period that indicates how often the task is executed, a worst-case execution time, and an assigned priority (Stankovic, 1998). These restrictions in the system's functionality affect the features of an agent that needs to be modeled as a RTS. The main problem is that if the tasks executed by the agent are not temporal bounded in the correct way, it is not possible to guarantee that the tasks will be completed before a given deadline. As scheduling a plan composed of these tasks is impossible, the reasoning process of agents that must work in real-time environments must therefore be temporal bounded. Agents in environments of this kind are typically referred to as real-time agents (RTA) (Julian & Botti, 2004; Sáenz et al., 2008). A RTA will be able to determine whether it has enough time to deliberate and to take into account the temporal cost of its cognitive reasoning process when it plans the execution of new tasks. If RTAs

* Corresponding author. Address: Faculty of Computer Sciences, Univ. Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain. Tel.: +34 923 294400; fax: +34 923 294514.
*E-mail addresses:* mnavarro@dsic.upv.es (M. Navarro), fcofds@usal.es (J.F. De Paz), vinglada@dsic.upv.es (V. Julián), srg@usal.es (S. Rodríguez), jbajope@usal.es (J. Bajo), corchado@usal.es (J.M. Corchado).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.01.119

are applied to monitor security related activities, it would be possible to make critical decisions that can affect the planning of activities in a real-time way.

This paper focuses on the problem of monitoring security guards and mobile robots, and task planning in industrial scenarios. It proposes an innovative MAS that incorporates a special type of intelligent real-time agent characterized by an internal structure that integrates a CBP-BDI (Corchado & Laza, 2003; Glez-Bedia & Corchado, 2002) (Case-Based Planning) (Glez-Bedia & Corchado, 2002; Spalazzi, 2001) (Beliefs, Desires, Intentions) (Bratman, 1987) model. This type of agent provides advanced reasoning abilities and resolves new problems by making use of past experiences. Moreover, the CBP system proposed in this study incorporates a sub-symbolic model, based on artificial neural networks (ANN), for resolving problems at a low level of detail, which allows advanced prediction capability. The main characteristic of the new agent presented in this paper is its capability to solve problems with temporal constraints. The temporal bounded version of the CBP-BDI agent presented in this paper incorporates a modified cycle of a classic CBR process that employs past experiences to resolve new time bounded problems.

Although the MAS proposed in this research was used to develop a functional demonstrator initially designed to schedule and supervise routes for security guards and mobile robots working in industrial environments, it can be easily extended to be applied in other industrial areas, which require temporal constraints. The work presented in this paper is an extension of a previously existing system. The previous system was aimed at monitoring workers, but did not take temporal constraints into account (De Paz, Rodríguez, Bajo, & Corchado, 2009). In addressing this point, we will focus on presenting a new temporal bounded mechanism, its capability for re-planning routes while complying with the temporal restrictions that are inherent to real time agents, and how this mechanism can improve overall system performance. The tracking system includes technologies for monitoring the surveillance routes based on QR-CODES and GPS.

Section 2 provides a detailed explanation of the new Temporal Bounded CBP model proposed in this work and its integration within a BDI agent model. Section 3 presents the multiagent system specifically designed to monitor industrial scenarios; Section 4 describes a case study in industrial scenarios; and finally, Section 5 reports the results and conclusions obtained after testing the proposed approach.

## 2. Temporal bounded case-based planning mechanism

The problem of generating plans or, more specifically, routes in industrial environments is a highly dynamic problem that requires intelligent systems with great capacity for learning and adaptation. The case-based reasoning (CBR) systems are based on a paradigm where past experiences are used to resolve new problems (Kolodner, 1993). This makes them very appropriate to be used in changing environments, since they are able to adapt themselves to changes in the environment using memories. However, the CBR paradigm does not take temporal restrictions into account and, if we want to use CBR techniques as a reasoning mechanism in real-time agents, it is necessary to adapt these techniques so that they can be executed guaranteeing real-time constraints. In these situations, CBR phases must be temporal bounded to ensure that solutions are produced on time. For this reason, the system proposed in this paper uses a Temporal-Bounded CBR (TB-CBR), which is a modification of the classic CBR cycle, in order to adapt it to real-time domains. This approach will allow for a more efficient execution of time management, according to the real-time agent's goals, and it facilitates the dynamic addition of new deliberative capabilities in agents.

More specifically, our model employs a temporal-bounded CBP approach as a deliberative engine for an agent. CBP (Bajo, De Paz, De Paz, & Corchado, 2009; Corchado & Laza, 2003) is a variation of CBR, which consists of the idea of planning as remembering (Glez-Bedia & Corchado, 2002). In CBP, the solution proposed to solve a given problem is a plan; this solution is generated by taking into account the plans applied to solve similar problems in the past. The problems and their corresponding plans are stored in a memory of plans. In addition to a specific problem with a specific solution, further information about how the plans have been derived is also stored.

It should be noted that the TB-CBP engine is integrated within the BDI deliberative model of an agent. This makes it possible to integrate both the symbolic and sub-symbolic models so that the BDI-based agents can decide what action to take at any given moment according to their objectives. The BDI model is used to formalize the problem, after which the reasoning cycle of the CBP is integrated with the BDI model in order to calculate the final solution according to past experiences. The terminology used for a BDI agent model (Bratman, 1987; Corchado & Laza, 2003) is as follows:

- The environment or world $M$ and the changes that are produced within it, are represented as a set of variables that influence a problem faced by the agent

$$M = \{\tau_1, \tau_2, \ldots, \tau_s\} \quad \text{with} \quad s < \infty \tag{1}$$

- The beliefs are vectors of some (or all) of the attributes of the world using a set of concrete values

$$B = \{b_i / b_i = \{\tau_1^i, \tau_2^i, \ldots, \tau_n^i\}, n \leqslant s \quad \forall i \in N\}_{i \in N} \subseteq M \tag{2}$$

- A state of the world $e_j \in E$ is represented for the agent by a set of beliefs that are true at a specific moment in time $t$. Let $E = \{e_j\}_{j \in N}$ set of status of the world. If we fix the value of $t$ then

$$e_j^t = \{b_1^{jt}, b_2^{jt}, \ldots b_r^{jt}\}_{r \in N} \subseteq B \quad \forall j, t \tag{3}$$

- The desires are the applications between the state of the current world and another that it is trying to reach

$$d : \underset{e_0}{E} \rightarrow \underset{e^*}{E} \tag{4}$$

- Intentions are the way that the agent's knowledge is used in order to reach its objectives. A desire is attainable if the application $i$ exists, as defined through $n$ beliefs:

$$i : \underset{(b_1, b_2, \ldots, b_n, e_0)}{BxBx\overset{n)}{\cdots}xBxE} \rightarrow \underset{e^*}{E} \tag{5}$$

- We define an agent action as the mechanism that provokes changes in the world making it change the state,

$$a_j : \underset{e_i}{E} \rightarrow \underset{a_j(e_i) = e_j}{E} \tag{6}$$

- Agent plan is the name we give to a sequence of actions that, from a current state $e_0$, define the path of states through which the agent passes in order to reach the other world state.

$$p_n : \underset{e_0}{E} \rightarrow \underset{p_n(e_0) = e_n}{E} \tag{7}$$

$$p_n(e_0) = e_n = a_n(e_{n-1}) = \cdots = (a_n \circ \cdots \circ a_1)(e_0) p_n \equiv a_n \circ \cdots \circ a_1$$

With respect to a CBP system, it can be defined as a system with the following components:

- A case base set ($\beta$). A case base $B \in \beta$ is a finite indexed set of cases. A case base is defined as a tuple where $(\{c1, c2, \ldots, cn\}, t)$. $\{c1, c2, \ldots, cn\}$ are the cases that compose the case base and $t$ is the finite set of characteristics that allow cases to be indexed.

- A case (c) represents a past experience. A case can be represented as a sequence of states from the environment: $c$ = {initial_state, {action_x[intermediate_state]}+, final_state}. Each state is represented through a set of attributes that define the environment where the CBP system is placed. The states are divided into three groups:
  - Set of initial states $\{st_i\}$, which represent the description of the problem to be solved.
  - Set of intermediate states $\{st_{inter\_i}\}$, which describe the different states the environment has gone through before the final state is achieved.
  - Set of final states $\{st_{fin\_i}\}$, which represent the description of the environment when the initial goals have been reached.

Moreover, a case contains actions (act), which represent the set of actions applied to each one of the states. The cycle of the CBP system is defined through the phases shown in Table 1. In this table, it is possible to appreciate how, in the retrieve phase, cases $c1, c2, \ldots, ck$, which have a problem description most similar to the current problem *stn*, are obtained from the case memory *B*. This is done employing a metric *A*. In the reuse phase an initial solution is obtained ($\{act\_ni, \{stinter\_ni\}*\}$ +, *stfinal_n*) from the previously retrieved cases and the problem description. In the revise phase the validity of the proposed solution is evaluated; finally, in the learning phase, the system can learn a new experience.

The way to integrate a CBP system in a BDI agent basically consists of defining a correspondence between the concept of case and the concepts managed by a BDI agent (beliefs, desires and intentions). Consequently, when a BDI agent needs to solve a new problem, it will use its beliefs, desires and intentions to build the new solutions. In this kind of system, these previous desires as well as the related beliefs and intentions, are stored as cases according to the following correspondence:

| **Case ⟨Problem, Solution, Result⟩** | **BDI Agent** |
|---|---|
| Problem: initial_state | Belief: state |
| Solution: sequence of ⟨action, [intermediate_state]⟩ | Intention: sequence of ⟨action⟩ |
| Result: final_state | Desire: set of ⟨final_state⟩ |

A belief is a state that can be *initial* (represents the problem that must be solved), *intermediate* (represents an intermediate state of the problem, problem that is passed before reaching the final state) or *final* (represents the obtained result starting from the initial state and executing a set of actions). Each belief has a set of attributes that describe it. A desire is a set of final states and, depending on the problem, can be formed by one or more final states that the agent wants to achieve. An intention is considered an ordered set of actions. Actions are operations that can be executed over a specific state. Each action is defined through a name and a set of arguments. With respect to the characteristics of the system, it may be necessary to add some additional beliefs to obtain a correct execution of the agent. These additional beliefs are actually indexes that allow the case memory to be organized. Moreover, similarity functions are added to determine the equality degree between two states.

**Table 1**
Cycle of a CBP system.

| | |
|---|---|
| Retrieve | $(c1, c2, \ldots, ck, A) \leftarrow$ Retrieve $(stn, B)$ with $ck = \{stk, \{act\_ki, \{stinter\_ki\}*\}+, stfin\_k\}$, $k > 0$ and $i > 0$ |
| Reuse | $(stn, \{act\_ni, \{stinter\_ni\}*\}+, stfinal\_n) \leftarrow$ Reuse $(stn, (c1, c2, \ldots, ck), A)$ |
| Revision | $(stn, \{act\_ni, \{stinter\_ni\}*\}+, stfinal\_n) \leftarrow$ Revision $(stfin\_n)$ |
| Retain | $(stn, \{act\_ni, \{stinter\_ni\}*\}+, stfinal\_n, B) \leftarrow$ Learning $(cn, B)$ |

This CBP system, which is integrated as a reasoning mechanism into a BDI agent, must be adapted, as previously stated, for its correct execution in real-time environments. Specifically, it is necessary to redefine the CBP cycle as a TB-CBP. In the TB-CBP proposed in this study, two main stages are defined: the learning stage, which consists of the revise and retain phases of the CBP cycle; and the deliberative stage, which includes the retrieve and reuse phases. The execution of both stages will be time scheduled. Therefore, the real-time agent has the ability to choose between either assigning more time to the deliberative stage or keeping more time for the learning stage, thus making agents more sensitive to updates. These new stages must be designed as an anytime algorithm (Dean and Boddy, 1988), where the process is iterative and each iteration is time-bounded and may improve the final response. The anytime behavior of the TB-CBP is achieved through the use of two loop control sequences. The loop condition is built using the *enoughTime* function, which determines if a new iteration is possible according to the total time that the TB-CBP has to complete each stage. The first phase of the algorithm executes the learning stage. This stage is executed only if the agent has the solutions from previous executions stored in the *solutionQueue*. The solutions are stored just after the end of the deliberative stage. The deliberative stage is only launched if the agent has a problem to solve in the *problemQueue*. This configuration allows the agent to launch the TB-CBP in order to only learn (no solution is needed and the agent has enough time to reason about previous decisions), only deliberate (there are no previous solutions to consider and there is a new problem to solve) or both. The following algorithm presents this adaptation of the CBP cycle:

The TB-CBP cycle starts at the learning stage, where it checks to see if there are previous cases waiting to be revised and possibly stored in the case-base. In our model, the plans provided at the end of the deliberative stage will be stored in a solution list while feedback about their utility is received. When each new TB-CBP cycle begins, this list is accessed. If there is enough time, the learning stage is implemented for those cases whose solution feedback has been recently received. If the list is empty, this process is omitted.

The next stage to be implemented is the deliberative stage. The retrieval algorithm is used to search the case-base and retrieve a case that is similar to the current case (i.e. one that characterizes the problem to be solved). Each time a similar case is found, it is sent to the reuse phase where it is transformed into a suitable plan for the current problem by using a Reuse algorithm. Therefore, at the end of each iteration of the deliberative stage, the TB-CBP method is able to provide a plan for the problem at hand, although this plan can be improved in subsequent iterations if the deliberative stage has enough time to perform them.

This section has presented an adaptation of the CBP cycle to be predictable and, consequently, ready for its employment in real-time environments. The next sections illustrate the proposal through the design of a MAS for industrial environments including agents with the proposed TB-CBP reasoning capability and, moreover, its application in a real case study.

## 3. Multiagent architecture for industrial environments

A multi-agent system was developed to provide control over the activities performed by the staff and robots responsible for overseeing the industrial environments. The agents in the system calculate the surveillance routes for the security guard robots according to the working shifts, the distance to be covered in the facilities, and the security guards and robots available. The system has the ability to re-plan the routes automatically according to the security personnel available. GPS and QR-CODES are key technologies in this

development. The system structure is defined by five different kinds of agents (De Paz et al., 2009):

- *Planner Agent.* Automatically generates the surveillance routes that are sent to the Manager Agent to distribute them among the security guards and robots.
- *Guard Agent.* Is associated to each Robot or mobile device. Manages the QR-CODES reader. Communicates with Controller Agents to check the completion of the assigned surveillance routes, to obtain new routes, and also to send the GPS position tags and QR-CODES information via Wi-Fi or HSDPA.
- *Manager Agent.* Controls the other agents in the system. Manages the connection and disconnection of Guard Agents to determine the number of security guards and mobile robots available. The information is sent to the Planner Agent to generate new surveillance routes.
- *Controller Agent.* Checks the control points to monitor security guards' activities.
- *Advisor Agent.* Administers the communication with the supervisors (person). Receives an incident report from the Manager Agent and decides if it is sent to the supervisor. Incidents can be sent via Wi-Fi, SMS or HSDPA.

### 3.1. Planner Agent

The most important agent in the system is the Planner Agent, which incorporates the TB-CBP-BDI model. The Planner Agent is modeled as a real-time agent to ensure that the plans made by the TB-CBP-BDI reasoning model are carried out within the specified time. This agent can be used for different purposes in industrial environments. In this case study we focused on the tasks performed by the security guards and mobile robots, but the proposed system could be easily extended to different types of workers. In order to adapt the TB-CBP-BDI model to the problem of security in industrial spaces, the environment equation (1) was defined through the following variables: security guards, coordinates for every control point, initial time, start time, deadline and service time. The current state (3) is obtained through the number of available security guards, their corresponding control points at that moment, and the time. The desires (4) are represented as the surveillance route that covers all the control points in the least amount of time given the temporal constraints. The intentions (5) are given for the neural networks that establish the sequence of states through which the system passes in order to reach the final state in which the surveillance routes have been successfully completed. Eqs. (9) and (10) show the structure for a plan (7).

The planning is carried out by two methods: the first is a simple method that obtains very quick, albeit low quality, results; the second uses a neural network based on the Kohonen Network (Leung, Jin, & Xu, 2004). This method needs more time to obtain results that vary in quality according to the time spent calculating them. Each of the phases of the TB-CBP-BDI planner is explained in detail in the following sub-sections:

#### 3.1.1. Learning stage (revise and retain)

When the security guards complete their rounds they provide a report indicating whether the route was completed correctly. The route is considered as successfully completed when it was finished on time. This information is stored in the Solution Queue.

At the beginning of the learning stage, the system must confirm if there are solutions in the Solution Queue. If there is still time to continue carrying out this stage, the *analysisResult* function will be applied to every solution found in the *SolutionQueue*. If this analysis indicates a positive assessment, then the complete plan is stored by the *retainResult* function. This plan contains the sequence of states and the corresponding belief value for each of them, i.e., the sequence of control points and their corresponding times. The number of replannings carried out determines the quality of the route. The information stored in the memory of plans follows the expressions (9) and (10).

If the problem includes temporal restrictions, this information is added to the rest of the plan information. The plan will therefore contain the following information:

$$\langle T = \{x_1, a_1, s_i, e_i, t_i)/x_i = (x_{i1}, x_{i2}), i = 1 \dots n\}, g\rangle \tag{8}$$

where $x_i$ position $(x,y)$ of every control point, $a_i$ arrival Time, $s_i$ initial time, $e_i$ final time, $t_i$ service time.

Both the *analysisResult* and the *retainResult* functions have a fixed asymptotic cost $O(1)$ and as a result, the execution time associated with each of the functions is predictable.

#### 3.1.2. Retrieve

In this phase the most similar plans resolved in the past, including all the control points indicated in the new problem, are recovered. The following record gives the information from the plan:

$$\langle T = \{t_i\}, g\rangle \quad i = 1 \dots n \tag{9}$$

$$t_i = (x_i, a_i)/x_i = (x_{i1}, x_{i2})$$

where $x_i$ the control point $i$ that will be visited, $(x_{i1},x_{i2})$ the coordinates of point $i$ and $g$ the number of security guards, $a_i$ arrival time. The initial state corresponds to the state in which the tasks are unsorted, and the final state contains all tasks sorted. The route $r_i$ is recovered following Eq. (10) by means of the search function:

$$r = \{R_l\} i = 1 \dots g \quad \text{where} \quad r_i \subseteq T, r_i \cap r_j = \phi \quad \forall i \neq j \quad j = 1 \dots g \tag{10}$$

where $R$ is the variable case shown in Algorithm 1. The search function retrieves the plans that contain the same control points, and selects the distribution that provides the minimum number of replanning actions.

**Algorithm 1**: Temporal-bounded CBP.

```
    Input  : t_max
    Output:
2.1  (t_learning, t_deliberative) ⟵ timeManager(t_max)
2.2  if solutionQueue ≠ ∅ then
2.3      while enoughTime(t_now, t_revise, t_retain, t_learning) and
         solutionQueue ≠ ∅ do
2.4          (st_n, {act_ni, {st_inter_ni}*}+, st_final_n) ⟵ pop(solutionQueue)
2.5          {adequate ⟵
             Revision(st_fin_n, (st_n, {act_ni, {st_inter_ni}*}+, st_final_n))}^{≤t_revise}
2.6          if adequate then
2.7              {Learning((st_n, {act_ni, {st_inter_ni}*}+, st_final_n), B)}^{≤t_retain}
2.8          end
2.9      end
2.10 end
2.11 if problemQueue ≠ ∅ then
2.12     problem ⟵ pop(problemQueue)
2.13     bestSolution = ∅
2.14     repeat
2.15         {(c_1, c_2, ..., c_k, A) ⟵ push(Retrieve(st_n, B))}^{≤t_retrieve}
2.16         {(st_n, {act_ni, {st_inter_ni}*}+, st_final_n) ⟵
             Reuse(st_n, (c_1, c_2, ..., c_k), A)
2.17         bestSolution ⟵
             bestSolution((st_n, {act_ni, {st_inter_ni}*}+, st_final_n), bestSolution)
             }^{≤t_reuse}
2.18     until ¬enoughTime(t_now, t_retrieve, t_reuse, t_deliberative) ;
2.19     solutionQueue ⟵ push(bestSolution)
2.20     return bestSolution
2.21 end
```

The time that the search function allocates for recovering the cases is limited by its asymptotic temporal cost $O(n)$, where $n$ is the number of cases stored in the database.

*3.1.3. Reuse*

In this phase, the retrieved routes are represented as cases and adapted to the temporal restrictions stated in the problem description. The process is carried out according to Algorithm 2.

**Algorithm 2**: Algorithm for the *adaptSolution* function.

```
    Input  : cases, problem
    Output: solution
2.1 if cases = ∅ then
2.2 |  cases ⟵ obtainSet (problem)
2.3 end
2.4 solutionLight ⟵ lightPlanner (cases)
2.5 while enoughTime(t_now,(t_deliberative − t_retrieve)) do
2.6 |  solutionHeavy ⟵ HeavyPlanner (cases)
2.7 end
2.8 solution ⟵ HighQuality (solutionHeavy,solutionLight)
2.9 return solution
```

In the first step, when no data has been retrieved in the recovery phase, Algorithm 2 uses the *obtainSet* function to generate a distribution of the control points that the security should visit. To do so, we use the $k$-means learning algorithm (Jennings & Wooldridge, 1998) to calculate the optimal routes and assign them to the available security guards. The inputs of the algorithm are $x_i \equiv (x_{i1}, x_{i2}) \quad i = 1, \ldots, N$, where $i$ represents the control point coordinates, $N$ the number of control points in the route, and $w_{kj}$ is the position of the centroid $k$ in the output layer that connects with the neuron $j$ in the input layer. Once the input and output are established, the modified $k$-means algorithm is carried out to create a new allocation:

- Establish the number $k$ of initial groups.
- Initiate the $k$ initial patterns. $w_{ij} = x_{ij}$.
- For each of the patterns, establish the nearest neuron of the output layer and associate the pattern with it. The Euclidean distance is used. $Q_k$ represents the set of input patterns associated with the neuron of the output layer $k$.

$$Q_k = \{x_i / d(w_k, X_i) \leq d(w_r, X_i) \; \forall k \neq r\} d(w_r, X_i) = \|w_r - X_i\| \quad (11)$$

- Calculate the new centroids of the neurons of the hidden layer as the average of the input associated patterns.

$$w_{kj} = \frac{1}{\#Q_k} \sum x_{sj} \quad \text{with} \quad x_s \in Q_k \quad (12)$$

- Repeat from step 3 until the modification of the centroids are less than $\alpha$ or until the maximum number of iterations is reached. It is necessary to establish a maximum number of iterations in order to shorten the maximum execution time.

$$\sum \Delta w_k = \sum \|w_k(t) - w_k(t-1)\| < \alpha \quad (13)$$

Once the points have been distributed among the different routes $r_i$, the TB-CBP-BDI starts spreading the control points among the available security guards.

There are exit methods that can calculate optimal routes, including but not limited to: genetic algorithms (GAs) (Rosenkrantz, Stearns, & Lewis, 1977), integer lineal programming (Dantzig, Fulkerson, & Johnson, 1954), Lin Kernighan Heuristic (LKH) (Lin & Kernighan, 1973), self-organizing maps (SOM) (Kohonen, 2001) neural network. However, it is difficult to take time restrictions into account in these heuristics, as only the GAs and SOM are easily adaptable to this situation. In this study, the optimal route for each guard is calculated using a modified SOM neural network, although a comparative study with the GAs remains pending.

In order to obtain a solution within the specified time $t_{reuse}$, different procedures were used to generate the plan. The first procedure (*lightPlanner*) can generate a low quality, predictable solution in low execution time, while the second (*heavyPlanner*) uses a SOM adapted to the *AnyTime* approach to generate priori solutions of superior quality (depending on the processing time of the method). Both systems employ automatic planning and select the best quality plan once the process is complete or the available time has expired. The quality is measured according to the final distance. The following section describes both processes.

*3.1.4. lightPlanner*

The algorithm used to calculate the route is very basic. It simply puts the control points in order according to the arrival time and then selects each of the points that are closest to the last control point visited, much like the nearest neighbor algorithm. The algorithm has an asymptotic cost of $O(n \log n)$ because the data must have already been put in order. As a result, the execution time can be reduced if the number of existing points is fixed.

*3.1.5. heavyPlanner*

The *heavyPlanner* function is a modified SOM that can calculate the routes that must satisfy certain temporal restrictions and must be calculated in a limited amount of time. The SOM has two layers: IN and OUT. The IN layer has two neurons, corresponding to the physical control point coordinates. The OUT layer has the same number of control points on each route (Martín et al., 2005). Given $x_i \equiv (x_{i1}, x_{i2}) \quad i = 1, \ldots, N$ the $i$ control point coordinates and $n_i \equiv (n_{i1}, n_{i2}) \quad i = 1, \ldots, N$ the $i$ neuron coordinates on $\Re^2$, and $N$ the number of control points in the route. The weight actualization formula is defined by the following equation:

$$w_{ki}(t+1) = w_{ki}(t) + \eta(t) g(k, h, t)(x_i(t) - w_{ki}(t)) \quad (14)$$

where $w_{ki}$ is the weight that connects the IN layer $i$ neuron with the OUT layer $k$ neuron, $t$ represents the interaction, $\eta(t)$ the learning rate decreasing with the number of iterations in order to stabilize the learning; and $g(k,h,t)$ the neighborhood function, which depends on three parameters: the winning neuron, the current neuron and the interaction. A decreasing neighborhood function is determined by the number of interactions and the winning neuron distance. To resolve optimization problems according to the temporal restrictions, it is necessary to modify the definition of the neighboring function. The restrictions that must be considered are: service time (the time needed by a security guard to check a control point), and the initial time and final time, which indicate the interval of time that the guard needs to arrive at the destination and check the control point. If the security guard arrives before the start time then he will wait. The coordinates have been scaled so that the distance travelled is also a unit. This is because it is necessary for the units to be comparable to the input layer of the ANN. The information available to the input layer will be: coordinates, start time, end time and service time.

The modification of the values corresponding to the weights of the links between neurons will be made in the same manner as with the previously explained network (14), defining a new neighborhood function. Moreover, a new distance function will be defined. It will be called temporal distance, and it replaces the previously used Euclidean distance in the neighborhood function. In order to establish the arrival time at the control point, it is necessary to take into account the space per time unit that a security guard employs in traveling from one control point to another. The new function defined is:

$$dt_{ij} \equiv dt(x_i, x_j) = Max\{f_{ij} + t_i, b_j\} \quad (15)$$

where $t_i$ is the accumulated time to arrive to control point $i$ plus the service time, $b_j$ the start time, $f_{ij}$ the distance between neurons $i$ and $j$.

Therefore the neighborhood function will be:

$$g(k,h,t) = Exp\left[\left(-\frac{|k-h|}{N/2}\right)\frac{df_{kh}}{Max_{ij}\{d^*_{ij\,i,j}\}} - \lambda\frac{|k-h|t}{\beta N}\right] \quad (16)$$

$$df_{kh} = \begin{cases} \sqrt{(n_{k1}-n_{h1})^2 + (n_{k2}-n_{h2})^2} & si \quad c_k - dt_{ok} < d^*_{kh} \\ 0 & eoc \end{cases} \quad (17)$$

where $\lambda$ and $\beta$ are determined empirically. The value of $\lambda$ is set to 1 by default, and the values of $\beta$ are set between 5 and 50, $t$ is the current interaction. $Exp[x] = e^x$, $N$ is the number of control points, $f_{ij}$ is the distance between two points $i$ and $j$, $n_{ij}$ coordinate $j$ of the neuron $i$, $d^*_{ij} = d^*(x_i, x_j) = f_{ij} + s_i$ with $s_j$ being the service time for the control point $j$, and $c_k$ being the closing time of the neuron $k$.

The use of the new distance $df_{kh}$ allows the neurons to be swapped with their neighbors if the temporal restrictions have not been met. However, this method does not guarantee that the system can achieve a valid solution.

The learning rate depends on the number of interactions, as can be seen in the following equation:

$$\eta(t) = Exp\left[-\sqrt[4]{\frac{t}{\beta N}}\right] \quad (18)$$

The neurons activation function is the identity. Having initially considered a high neighborhood radius, the weights modifications affect the nearest neurons. Reducing the neighborhood radius, the number of neurons affected decreases, until just the winning neuron is affected.

The process concludes when one of the following conditions is satisfied: there is only one neuron associated with each control point, or the maximum processing time allowed was reached without having obtained a complete solution. In the latter case, the points are reviewed in order and are associated to the nearest neuron. To determine the optimal route, the $i$ neuron is associated with the $i+1$ neuron, from $i = 1,2,\ldots,N$, covering all the neurons vectors. Finally, the order of the routes is validated and the order of the neurons is exchanged in case they do not meet the defined time intervals.

$$t_{reuse} \geqslant t_{obtained} + t_{lightPlanner} + t_{heavyPlanner} \quad (19)$$

where $t_{obtained}$ is the execution time obtained from *obtainedSet*, which can have two values: 0 if there is a previous plan or cost $O(n)$ if it must perform a distribution. $t_{lightPlanner}$ will have cost $O(n * \log n)$, where in both cases $n$ are the control points. The temporal cost of executing *heavyPlanner* ($t_{heavyPlanner}$) will be variable, located within the interval $0 \leqslant t_{heavyPlanner} \leqslant (t_{reuse} - (t_{obtained} + t_{lightPlanner}))$.

## 4. Case study: monitoring surveillance routes in a industrial scenario

In order to evaluate the multi-agent system proposed in Section 3, a case study in a real scenario was developed to provide security in an industrial environment in the Castilla y León region of Spain. In this scenario, it is necessary to establish surveillance routes in order to guarantee security, especially during the night. Security was provided by means of security guards that complete static routes and check the safety of the environment. However, the routes were static and most of the times were not completed by the security guards. In order to resolve these problems, a multiagent system was developed to generate dynamic routes and checkpoints, and ensure the completion of the surveillance. Moreover, the proposed solution incorporates mobile robots that can automatically monitor certain areas when the security guards are not available.

This section presents an example of an execution of the multiagent system, which serves to illustrate the generation and completion of plans, as well as the information shown to security guards on their mobile devices. Different tools were used to develop the case study, including real-time operating systems to control the surveillance robots. The application for the mobile devices was implemented using *android*, but it can be easily extended to other operating systems. Android was chosen because of its particular capabilities to integrate the surveillance facility within existing technologies and applications. The Google maps library was used to represent the surveillance environments, providing on-line update capability and an easy adaptation to alternative environments. Moreover, the Google zxing project was used to manage the codes in *android* devices. The barcode scanner was used to implement the code's reader.

The scenario was tested with nine security guards and two mobile robots. The mobile robots are used in areas of difficult accessibility and require real-time control. There were 39 checkpoints in total. The process begins with the generation of routes and the assignment for each security guard. As can be seen in Fig. 1a, and b, security guard 1 receives a schedule containing his personal route. The guard can see a map with the assigned route and the checkpoints that need to be visited, and the imposed temporal restrictions. From the point of view of the guard, the route is composed of a series of stages that need to be completed.

As shown in Fig. 2a–c, once the security guard achieves a checkpoint, he uses the QR-CODES or GPS available on his mobile device to inform the multiagent system about his actions. Fig. 2a shows the option selected by the security guard to assess the checkpoints. Fig. 2b shows an example of a QR-CODE scan after pushing the check point button in Fig. 1a. Once the code has been scanned, the Security Guard agent interprets the text registered in the QR-CODE and this information is sent to the Manager Agent. As shown in Fig. 2c, the information shown to the security guard consists of the name of the control point together with the *id* of the checkpoint. The Planner Agent can then check the status of the execution of the plans (routes) and make decisions accordingly. If there is a delay or an incident, the Planner Agent can perform an automatic re-planning of the assigned tasks.

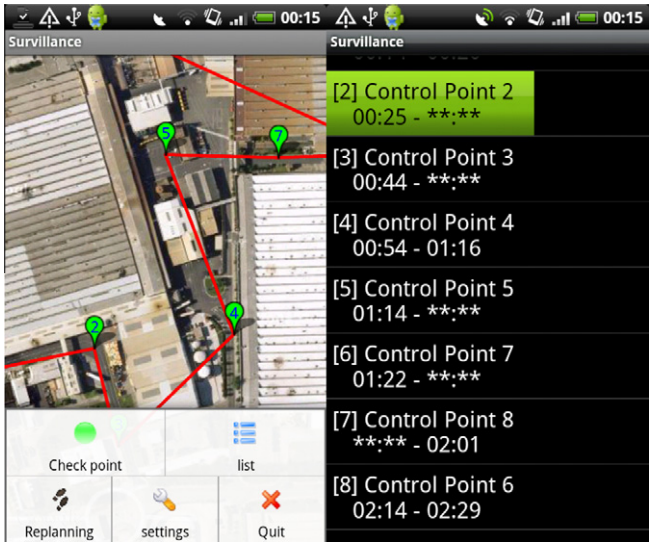


**Fig. 1.** (a) Calculated route and (b) list of checkpoint and their corresponding temporal constraints.

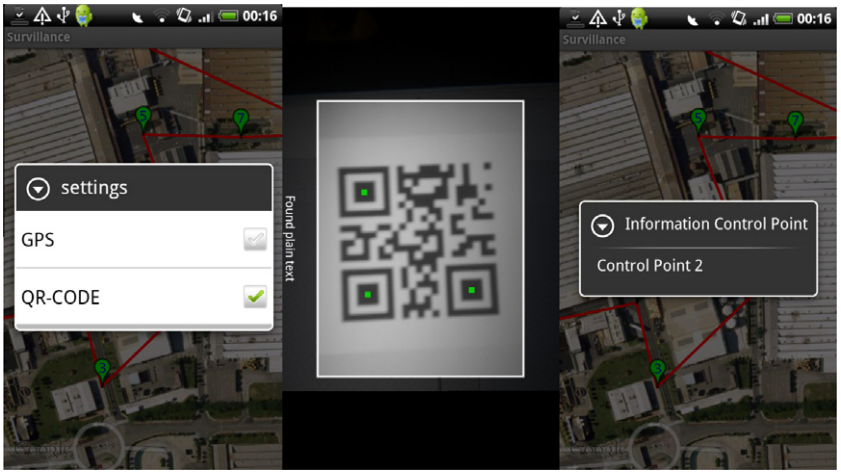


**Fig. 2.** (a) Selection of the tracking mechanism, (b) example of scanning of a checkpoint, and (c) information obtained from the checkpoint.

**Table 2**
Execution times for the different system functions.

| Functions | Asintotic | 20 | | 30 | | 40 | | 50 | | 60 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cost | $\bar{x}$ | wcet | $\bar{x}$ | wcet | $\bar{x}$ | wcet | $\bar{x}$ | wcet | $\bar{x}$ | wcet |
| analysesResult | $O(1)$ | >1 | 1 | >1 | 1 | >1 | 1 | >1 | 1 | >1 | 1 |
| retainResult | $O(1)$ | 201 | 307 | 224 | 307 | 217 | 307 | 232 | 307 | 224 | 307 |
| Search | $O(n)$ | 271 | 473 | 268 | 473 | 283 | 473 | 265 | 473 | 221 | 473 |
| *adaptSolution* | | | | | | | | | | | |
| obtainSet | $O(m)$ | 0.006 | 0.02 | 0.008 | 0.02 | 0.010 | 0.03 | 0.011 | 0.03 | 0.013 | 0.03 |
| lightPlanner | $O(m \log m)$ | 0.001 | 0.1 | 0.002 | 0.01 | 0.003 | 0.01 | 0.003 | 0.015 | 0.004 | 0.015 |
| heavyPlanner | – | 1114 | | 2416 | | 4490 | | 7322 | | 11242 | |

**Table 3**
Route followed by a security guard under 10 time restrictions.

| C.P. | Position | Distance | Arrival | I.T. | End time | S.T. |
|---|---|---|---|---|---|---|
| 0 | (400–500) | 206 | 0 | 0 | 33,900 | 0 |
| 20 | (200–550) | 50 | 206 | 630 | 7030 | 10 |
| 22 | (200–500) | 36 | 690 | 1580 | 7980 | 10 |
| 24 | (230–520) | 50 | 1626 | 170 | 6570 | 10 |
| 27 | (280–520) | 28 | 1686 | 120 | 6520 | 20 |
| 29 | (300–500) | 50 | 1734 | 100 | 6500 | 10 |
| 30 | (250–500) | 120 | 1794 | 150 | 6550 | 10 |
| 6 | (160–420) | 63 | 1925 | 2570 | 8970 | 20 |
| 32 | (100–400) | 20 | 2653 | 3530 | 9930 | 30 |
| 33 | (80–400) | 54 | 3580 | 4450 | 10,850 | 40 |
| 31 | (100–350) | 50 | 4544 | 5410 | 11,810 | 20 |
| 35 | (50–350) | 58 | 5480 | 6360 | 12,760 | 10 |
| 37 | (20–400) | 20 | 6428 | 7310 | 13,710 | 20 |
| 38 | (0–400) | 50 | 7350 | 8230 | 14,630 | 30 |
| 39 | (0–450) | 50 | 8310 | 9180 | 15,580 | 20 |
| 36 | (50–450) | 30 | 9250 | 10,130 | 16,530 | 10 |
| 34 | (80–450) | 108 | 10,170 | 11,060 | 17,460 | 20 |
| 28 | (40–550) | 81 | 11,188 | 12,070 | 18,470 | 20 |
| 26 | (80–620) | 72 | 12,171 | 13,050 | 19,450 | 10 |
| 23 | (140–660) | 305 | 13,132 | 14,030 | 20,430 | 10 |
| | | 1502 | 14,345 | | | |

There are a number of zones that the human guards are unable to access due to a variety of reasons such as a narrow roads, low roofs, dangerous areas, etc. Mobile robots were therefore used to navigate through these difficult zones. The mobile robot model selected in this study to carry out this function is the Pioneer P3-AT. This model offers an interesting embedded computer option, which allows for the possibility of onboard vision processing, in addition to Ethernet-based communications, laser, DGPS, and other autonomous functions. The P3-AT carries up to three hot swappable batteries, and optional eight forward and eight rear sonar sense

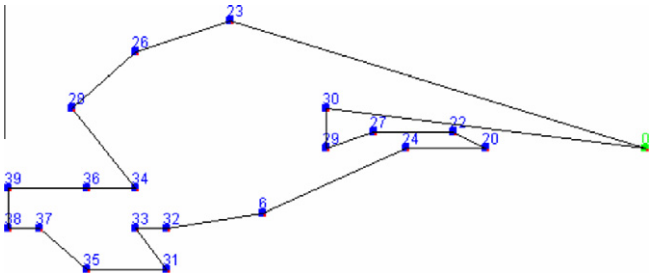


**Fig. 3.** Route calculated from Table 3.

obstacles from 15 cm to 7 m in length. The P3-AT's powerful motors and its four knobby wheels can reach speeds of .8 m per second and carry a payload of up to 12 kg. The P3-AT was updated with an indoor PTZ camera system and a speech and audio package that permit it to capture information from the environment.

The main function of the mobile robots is to navigate by following the route proposed by the Planner Agent. While the robot is moving it can capture images and sounds from the road. The obtained information will be analyzed by an expert to determine whether any anomaly in the environment exists.

The next section presents the results and conclusions obtained after having tested the proposed multiagent system in different scenarios.

## 5. Results and conclusions

The system presented in this paper was implemented and tested in experimental and controlled scenarios and under real time restrictions. The tests included various simulations with varying configurations of the control points and a fixed number of security

guards. This allowed us to compare the behavior of each of the algorithms and to estimate the execution time. Table 2 shows the results obtained for the different configurations, with the case memory limited to 50 plans. Each of the different functions was executed 1000 times with the following results: where $n$ is the number of elements in the data base, $m$ is the number of control points. All of the times are given in nanoseconds, except for those in the *adaptSolution*, which are milliseconds.

In a first step, the operation of the sub-symbolic model applied in the reuse phase of TB-CBP-BDI was checked. The model was implemented through sub-symbolic ANNs. Table 3 represents the plans scheduled by the neural network after the checkpoints had been divided. Table 3 gives a description of a sample surveillance route for a security guard. It also identifies the control point (CP) to visit, the location of the control point (coordinates), the distance between the current and the next control point, the accumulated time from the initial control point, the start time (IT), which represents the earliest time to check the control point (i.e., the control point can't be checked prior to this time), the end time (FT), which represents the maximum time allowed for the arrival, and finally the service time (ST). The default upper limit is comprised of half of the working shift, 14,400 ($4 * 3600$). To simplify the results, we established that the speed at which the guards move is 1 m/s. The final distance obtained is 1502 m. The result obtained by *lightPlanner* prior to applying *heavyPlanner* was 2237 m. Fig. 3 provides a graphical representation of the route calculated by *heavyPlanner*.

For all of the tests performed, the real time agent was capable of finding a plan within the deadline assigned for the task. This time fluctuated within a range of values that included the minimum time required to execute the TB-CBP-BDI algorithm, in which the *heavyPlanner* function is not executed, and a value greater than the maximum execution time provided by the *heavyPlanner*. Lesser times are disregarded because the execution time would be greater than the imposed time restrictions.

It is possible to determine the number of security guards needed to cover an entire area with routes so that human resources are optimized. In addition, the system provides the supervisors with relevant information to monitor the workerś activities, and to detect incidents in the surveillance routes automatically and in real time. The system presented in this study can be easily adapted to other categories of workers and other scenarios with similar characteristics. In this way, we believe that our approach can be very useful in industrial and manufacturing scenarios, where scheduling work shifts and accomplishing tasks are critical factors for improving the performance of the overall system. This approach may be a very appropriate application for environments where automatic re-planning is required and temporal constraints, such as robotic systems, reconfigurable factories or virtual organizations, are imposed. That is our next challenge.

## Acknowledgments

This development was supported by the JCYL SA071A08 project. The Spanish government (TIN2009-13839-C03), FEDER and CONSOLIDER-INGENIO (2010 CSD2007-00022).

## References

Bajo, J., De Paz, J. F., De Paz, Y., & Corchado, J. M. (2009). Integrating case-based planning and RPTW neural networks to construct an intelligent environment for health care. *Expert Systems with Applications, 36*(3–2), 5844–5858.

Bratman, M. (1987). *Intention, plans and practical reason*. Center for the Study of Language and Inf.

Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M., & Botti, V. (2008). Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications, 34*(1), 2–17.

Corchado, J. M., & Laza, R. (2003). Constructing deliberative agents with case-based reasoning technology. *International Journal of Intelligent Systems, 18*(12), 1227–1241.

Dantzig, G. B., Fulkerson, D. R., & Johnson, S. M. (1954). Solution of a large-scale traveling-salesman problem. *Operations Research, 2*, 393–410.

De Paz, J. F., Rodríguez, S., Bajo, J., & Corchado, J. M. (2009). Mathematical model for dynamic case based planning. *International Journal of Computer Mathematics, 86*, 1719–1730.

Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the 7th national conference on artificial intelligence* (pp. 49–54).

Glez-Bedia, M., & Corchado, J. M. (2002). A planning strategy based on variational calculus for deliberative agents. *Computing and Information Systems Journal, 10*(1), 2–14.

Inology (2005). Press note, June 9th. Available from: <http://www.controldetiempos.com/sala_de_prensa.htm#absentismo>.

Jennings, N., & Wooldridge, M. (1998). *Applications of intelligent agents*. Queen Mary & Westfield College: University of London.

Julian, V., & Botti, V. (2004). Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering, 11*, 135–149.

Kohonen, T. (2001). *Self-organising maps*. Springer-Verlag.

Kolodner, J. (1993). *Case-based reasoning*. New York: Morgan Kaufmann.

Leung, K. S., Jin, H. D., & Xu, Z. B. (2004). An expanding self-organizing neural network for the traveling salesman problem. *Neurocomputing, 62*, 267–292.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*, 498–516.

Martín, Q., Santos, M. T., & Paz, de Y. (2005). *Operations research: Resolute problems and exercises*. Pearson.

Partalas, I., Feneris, I., & Vlahavas, I. (2008). A hybrid multiagent reinforcement learning approach using strategies and fusion. *International Journal of Artificial Intelligence Tools, 17*(5), 945–962.

Rosenkrantz, D. E., Stearns, R. E., & Lewis, P. M. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing, 6*(3), 563–581.

Sáenz, E., Aztiria, A., García, C., Arana, N., Izaguirre, A., & Fillatreau, P. (2008). Forming processes control by means of artificial intelligence techniques. *Robotics and Computer-Integrated Manufacturing, 24*(6), 773–779.

Spalazzi, L. (2001). A survey on case-based planning. *Artificial Intelligence Review, 16*, 3–36.

Stankovic, J. A. (1998). Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer, 21*(10), 10–19.

Wooldridge, M., & Jennings, N. R. (1995). Agent theories, architectures, and languages: A survey. *Intelligent Agents*, 1–22.

Zheng, Y., Wang, J., & Xue, J. (2009). A team based supply chain management agent architecture. *International Journal of Artificial Intelligence Tools, 18*(6), 801–827.