Supporting Decision-Making for City Management through Automated Planning and Execution

Riccardo De Benedictis[0000−0003−2344−4088], Gloria Beraldo[0000−0001−8937−9739], Amedeo Cesta[0000−0002−0703−9122], and Gabriella Cortellessa[0000−0002−9835−1575]

Institute of Cognitive Sciences and Technologies (ISTC) - National Research Council of Italy (CNR) name.surname@cnr.it

Abstract. Urban intelligence is an emerging research field that aims at investigating the use of advanced technologies and data analysis techniques to enhance the efficiency, sustainability, and livability of urban areas. One of the components of urban intelligence is decision support which, among the possible implementations, can make use of forms of automated reasoning capable of *planning* the activities that must be carried out on the territory and, at the same time, of *reacting* to its dynamic evolution. Taking inspiration from the dual-process cognitive theories, this paper aims at investigating the integration of automated planning and rule-based systems as a means of supporting decision-making processes in urban management.

Keywords: Urban Intelligence · Planning · Decision Support System.

1 Introduction

Cities are complex and dynamic environments that require continuous management to ensure their efficiency, sustainability, and livability. As urban populations grow and become more diverse, the challenges of urban management become increasingly complex [16, 19]. In recent years, there has been a growing interest in the use of advanced technologies and data analysis techniques to support decision-making processes in urban management. Urban intelligence, in particular, is an emerging dynamic and evolving field of research that seeks to leverage advanced technologies and data analysis techniques to enhance the efficiency, sustainability, and livability of urban areas [3].

A critical component of the urban intelligence is the support to decisions which enables planners and policymakers to make informed decisions about urban development and management. A Cognitive Decision Support System (CDSS), in particular, is a type of Decision Support System (DSS) that incorporates cognitive theories and models of human decision-making to assist human decision-makers in complex decision-making tasks [18]. Among the available cognitive theories, the dual processing theory [15] has been widely applied

in the development of CDSSs [2, 22]. This cognitive theory posits that human cognition operates via two distinct systems or processes, often referred to as System 1 and System 2. More specifically, System 1 refers to fast, automatic, and intuitive thinking that relies on heuristics, mental shortcuts, and previous experience to make decisions or judgments. This system is often associated with our unconscious or intuitive mind and is involved in activities such as perception, pattern recognition, and emotional responses. System 2, on the other hand, refers to slower, deliberate, and analytical thinking that relies on logic, reasoning, and conscious effort to make decisions or judgments. System 2 is often associated with our conscious or rational mind and is involved in activities such as problem-solving, planning, and decision-making.

Taking inspiration from the dual processing theory, this paper presents a cognitive architecture, called COCO (from COmbined deduction and abduCtiOn logic reasoner), that aims at enhancing decision-making in urban management by combining a rule-based system to mimic the behavior of System 1, and a timeline-based planner, extended with semantic reasoning capabilities, to mimic the behavior of System 2. Section 2 provides background information on these two technologies, while Section 3 describes the COCO cognitive architecture, which combines a rule-based system and a timeline-based planner. This integration enables the planning of activities over extended time horizons, their execution, dynamic adaptation, and adaptive responses to changes in the urban environment. To demonstrate the effectiveness of this approach, Section 4 presents a case study in the city of Matera and provides some results. Finally, Section 5 summarizes the key findings and discusses future research directions.

2 Technical Background

This section provides some technical background on the two main components of the proposed approach for urban intelligence: rule-based systems and timeline-based planning. It explains how rule-based systems use a set of rules to make decisions and how they can efficiently react to new information. It also describes how timeline-based planning generates goal-oriented behaviors and is suitable for managing temporal information. Understanding these two components is crucial to understanding how they can be integrated to provide a more robust and flexible approach to urban intelligence.

2.1 Rule-based Systems

Rule-based systems are a type of Artificial Intelligence (AI) systems that use a set of "if-then" rules to make decisions or draw conclusions [14, 13]. These rules typically take the form of logical statements or condition-action pairs, where the condition is a set of input variables and the action is a set of output actions or conclusions. A knowledge base stores these rules along with some facts which are known to be true. The introduction of new facts triggers a reasoning

engine which, by applying the rules to the input data or situations, selects the appropriate actions or conclusions.

An example of a rule, for such a system, can be "IF the temperature is above 30 degrees Celsius AND the humidity is above 70%, THEN activate the sprinkler system in the park". In this example, the rule-based system is programmed to respond to specific environmental conditions (high temperature and humidity) by triggering an action (activating the sprinkler system) to keep lush the vegetation that inhabits the park. By introducing a fact stating that "the temperature is currently 35 degrees Celsius", and a fact stating that "the humidity is currently 75%", the previous rule is activated and the corresponding action is executed. It is worth noticing that when an action is executed, it can generate new facts or modify the existing ones, which can be used in subsequent reasoning and decision-making processes.

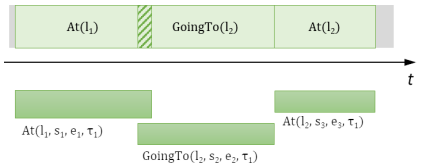
Rule-based systems can be particularly valuable in situations where decisions need to be made quickly and reliably. By using a set of pre-defined rules, decision-makers can quickly assess and respond to different situations, without the need for extensive analysis or deliberation. Furthermore, since the knowledge and reasoning process is based on explicitly defined rules, it is easy to understand how a rule-based system arrives at a certain decision or output. This is particularly important in the context of urban intelligence, where decision-makers and stakeholders need to have a clear understanding of how and why certain decisions are made.

2.2 Timeline-based Planning

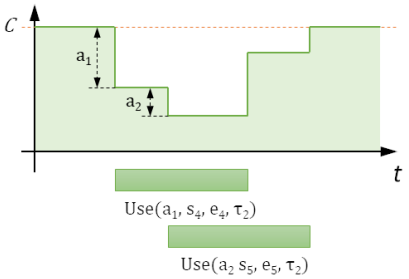
Automated planning [12] is another branch of AI that deals with creating computer programs that can generate plans, schedules, and strategies for accomplishing specific goals or tasks. Timeline-based planning [17], also known as constraint-based planning, is a type of automated planning where activities are organized over timelines and scheduled based on their temporal constraints and dependencies. This paper extends the formalization defined in [5] to handle also information not strictly tied to a specific time or time interval.

The basic building block of timeline-based planning, specifically, is the *token* which, intuitively, is used to represent the single unit of information. Through their introduction and their constraining during the planning process, tokens allow to represent the different components of the high-level plans. In its most general form, a token is formally described by an expression like $n\,(x_0, \ldots, x_i)_\chi$. In particular, n is a *predicate* symbol, $x_0, \ldots, x_i$ are its *parameters* (i.e., constants, numeric variables or object variables) and $\chi \in \{f, g\}$ is a constant representing the class of the token (i.e., either a *fact* or a *goal*).

The token's parameters are constituted, in general, by the variables of a *constraint network* $\mathcal{N}$ (refer to [6] for further details) and can be used, among other things, to represent temporal information such as the start or the end of some tasks. The semantics of the χ constant, on the contrary, is borrowed from Constraint Logic Programming (CLP) [1]. Specifically, while the facts are considered inherently true, the goals must be achieved as defined by a set of



(a) An *inconsistent* state-variable timeline. The first *At* token and the *GoingTo* token are temporally overlapping. The inconsistency can be removed, for example, by introducing a $e_1 \leq s_2$ constraint.



(b) A *consistent* reusable-resource timeline. The overlap of tokens is allowed as long as the simultaneous use of the resource is less than its capacity.

Fig. 1: Different timelines extracted by their associated tokens.

rules. Rules, in particular, are expressions of the form $n\,(x_0, \ldots, x_k) \leftarrow \mathbf{r}$ where $n\,(x_0, \ldots, x_k)$ is the *head* of the rule and $\mathbf{r}$ is the *body* of the rule. In particular, $\mathbf{r}$ represents the *requirement* for achieving any goal having the "form" of the head of the rule. Such requirements can be either a token, a *constraint* among tokens (possibly including the $x_0, \ldots, x_k$ variables), a *conjunction* of requirements or a (priced) *disjunction* of requirements. It is worth noting the recursive definition of requirement, which allows the definition of the body of a rule as any logical combination of tokens and constraints. To illustrate, let's consider a rule that outlines the required steps for installing an optical fiber line in a specific road.

$$OpticalFiber\,(r, s, e) \leftarrow \left\{ \begin{array}{c} [e - s \geq 20] \wedge \\ Trench\,(r_1 : r, s_1, e_1)_g \wedge \\ [s - e_1 \leq 7] \wedge [e_1 \leq s] \wedge \\ Repair\,(r_2 : r, s_2, e_2)_g \wedge \\ [s_2 - e \leq 8] \wedge [s_2 \geq e] \end{array} \right\}$$

According to the previous rule, the installation process takes a minimum of 20 time units. However, prior to commencing the installation, it is essential to excavate a trench along the road. The time between trench excavation and installation should not exceed 7 time units. Lastly, once the optical fiber is installed, the trench must be filled within a maximum of 8 time units to restore the road surface. The Trench and Repair predicates will share similar rules that establish the necessary conditions to accomplish the goals defined in the rule's body.

Similarly to CLP, through the application of the rules it is hence possible to establish and generate relationships among tokens. Compared to CLP, however, timelines introduce an added value: tokens may be equipped with a special object variable τ that identifies the *timeline* affected by the token. Different tokens with the same value for the τ parameter, in particular, affect the same timeline and, depending on the nature of the timeline, might interact with each other. There can be, indeed, different types of timelines. In case of *state-variable* timelines (see Figure 1a), for example, different tokens on the same state-variable cannot

temporally overlap. In case of *reusable-resource* timelines (see Figure 1b), on the contrary, tokens represent resource usages and can, hence, overlap as long as the concurrent uses remain below the resource's capacity. In this context, timelines can be viewed as a *global constraint* (see, for example, [6]) imposed on the tokens applied to them.

Given the ingredients mentioned above, we can now formally introduce the addressed planning problem. A *timeline-based planning problem*, specifically, is a triple $\mathcal{P} = (\mathbf{O}, \mathcal{R}, \mathbf{r})$, where $\mathbf{O}$ is a set of typed objects, needed for instantiating the initial domains of the constraint network variables and, consequently, the tokens' parameters, $\mathcal{R}$ is a set of rules and $\mathbf{r}$ is the requirement that needs to be satisfied so that the plan achieves the desired objectives. A *solution* is a set of tokens whose parameters assume values so as to guarantee the satisfaction of all the constraints imposed by the problem's requirement, by the application of the rules, as well as by the global constraints imposed by the timelines.

3 Thinking, Fast and Slow, Logically: COCO

In a rule-based system, the inference engine typically applies a set of production rules to a knowledge base in order to *deduce* new information or actions. The rules are usually written in a forward-chaining form, meaning that they are triggered when certain conditions (i.e., antecedents) are met, and then generate new information or actions (i.e., consequents). The search limited to match the rule conditions, enhanced with indexing techniques [8], combined with the almost total absence of backtracking, makes these approaches particularly efficient in reacting to new information. In timeline-based planning, on the contrary, the system works by searching backwards (*abduction*), from goals, to find a sequence of actions that can achieve that goals. This process involves constructing a plan by recursively applying rules and constraints, searching for valid solutions. When constraints do not propagate, the system backtracks, making the search for a solution more onerous from a complexity point of view. These approache are, nonetheless, specifically designed to handle temporal information and are more suited for generating goal-oriented behaviors that need to be executed within a certain time frame.

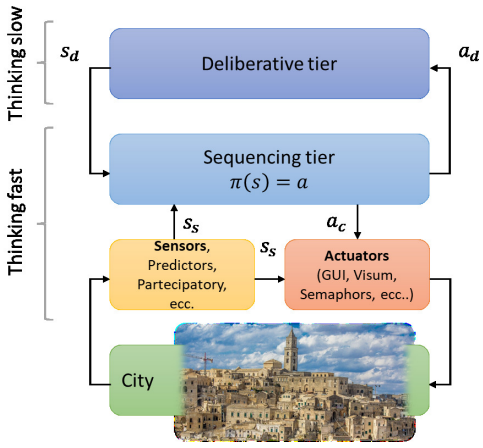


Fig. 2: The COCO three-layer architecture.

The proposed approach, depicted in Figure 2, aims to exploit the high-reactivity and explainability of the rule-based system, together with the powerful

planning capabilities of the timeline-based planner, to enable effective decision-making in dynamic and complex urban environments. Taking inspiration from classical robotics architectures [9], specifically, the COmbined deduction and abduCtiOn logic reasoner (COCO) consists of a *deliberative* tier responsible, through a timeline-based planner, for the generation, the execution and the dynamic adaptation of the plans; a *sequencing* tier which, through the CLIPS[1] rule-based system, executes a sequence of actions according to the current state of the world; and a *sensing* and a *controlling* tier, which respectively interprets data produced by sensors and translates the sequencer's actions into lower level commands for the actuators. The System 1 side is covered by the sequencing tier, which quickly recognizes and responds to familiar patterns and situations, generating abstractions from sensory data and low-level commands for actuators, functioning like automatic, intuitive decision-making. In contrast, the System 2 side is covered by the deliberative tier, which uses semantic and causal reasoning, and logical and arithmetic approaches to generate and adapt high-level plans based on dynamic environmental information.

The COCO state, according to which actions are selected from the sequencer tier, is represented through a set of facts in the rule-based system. Adding, modifying or deleting facts entails the execution of the actions through the activation of the rules. The facts, representing the state, are described by a combination of two distinct sets:

- the s_s set, containing facts generated by the sensing tier through a REST API, characterizes the consequences of the interpretation of sensory data, representing, for example, temperature, humidity, air quality, flows of vehicles on roads, etc.;
- the s_d set, containing facts generated by the deliberative tier, representing the high-level commands produced as a result of the execution of the planned tasks.

Similarly, the actions executed by the sequencer tier can be of two distinct types:

- the a_c actions, towards the controllers, responsible for various tasks, such as directly performing actions (e.g., activating a sprinkler) or indirectly influencing the city (e.g., communicating with municipal technicians or decision-makers);
- the a_d actions, towards the deliberative tier, responsible, for example, for the creation of the planning problems and for the execution and dynamic adaptation of the generated plans.

Notably, the sequencing tier, using the $\pi(s)$ policy, can act on the environment through a_c actions and introspectively on higher-level reasoning through a_d actions. The higher-level tasks (a.k.a. intrinsic motivations [21]) generated by the deliberative tier during plan execution are only one factor influencing

[1] https://clipsrules.net

User-defined function	Description
`new_solver(purpose, files)`	Creates a new solver for the given purpose and starts solving the planning problem contained in the given files.
`start_execution(solver_id)`	Starts the execution of the plan generated by the solver with the given ID.
`delay_task(task_id, delay_time)`	Delays the start of the task with the given ID by the `delay_time` amount of time.
`extend_task(task_id, extend_time)`	Extends the duration of the task with the given ID by the `extend_time` amount of time.
`failure(task_ids)`	Notifies the deliberative tier that the execution of the given set of tasks is failed. The executing plan should be adapted considering that the consequences of the failed tasks will no more be available.
`adapt(solver_id, files)`	Adapts the given plan by introducing new requirements (e.g., new goals).
`delete_solver(solver_id)`	Deletes the given solver and, if present, the corresponding plan.

Table 1: User-defined functions for interacting with the deliberative tier. These functions can be invoked, if necessary, by the sequencing tier.

the sequencing tier's actions. These tasks are not mandatory for the system's autonomy but serve as suggestions for the agent on what to do.

The sequencing tier of the COCO architecture is equipped with several actions that enable it to formulate the planning problem, execute and modify the solutions generated by the planner in a dynamic manner. In particular, the CLIPS system has been enhanced to support user-defined functions, summarized in Table 1, within rule-based system rules. One of these functions is `new_solver`, which creates a new solver from a string indicating the solver's purpose and a set of files comprising the planning model definition and the problem instance. For timeline-based planning, these files contain the rule definitions and the initial problem requirement. Upon invocation, this function generates a new `solver` fact (refer to Table 2 for the facts related to planning that have been considered into the knowledge base) in the knowledge base, which includes the solver's ID, purpose, and state (initially in the `reasoning` state until a solution is found). Notably, the sequencing tier remains active during the planning process, reacting to external inputs and taking into account the planner's reasoning status as necessary.

After the planning process is completed, the `solver` fact is updated to indicate that the planner is now in an `idle` state. At this point, a rule on the sequencing tier, that has a premise with a `solver` fact having the same purpose of the planner and the `idle` state, is used to trigger the execution of the plan through the user-defined function `start_execution`. The presence of this rule enables the option to postpone plan execution until additional conditions arise, if deemed necessary. The planner then modifies the `solver` fact by putting it in the `executing` state and starts executing the plan by sending tasks to the se-

Fact	Description
`solver(id, purpose, state)`	Declares the presence of a solver with a particular ID, purpose, and status. The status can be either `reasoning`, `idle`, `executing`, `adapting`, `finished` or `failed`.
`task(solver_id, id, type, pars, vals)`	Declares that a task is currently executing. The `solver_id` and `id` parameters indicate, respectively, the ID of the solver and the ID of the task. The `type` parameter indicates the type of the task (i.e., the n predicate symbol of the corresponding token). The `pars` and `vals` parameters indicate, respectively, the parameter names of the task and their values (i.e., the $x_0, \ldots, x_i$ names and values of the corresponding token).

Table 2: Facts asserted and modified during the execution of the plans. The rule-based system reacts to the presence of these facts as to the presence of facts asserted as a consequence of changes in the environment.

quencing tier in a timely manner. Aside from executing the plan, the sequencing tier can also perform other actions to modify the running plan. For instance, the `delay_task` and `extend_task` functions can request the delay of the start or end time of a task, respectively. The `failure` function can remove an activity from the plan, considering its effects on the execution of future activities. Lastly, the `adapt` function can introduce new requirements or goals within the current plan. As these adaptations can be quite expensive, effective strategies have been adopted to manage them [4, 11].

Reasoning on delays and failures in the sequencing tier results in the modification of the `solver` fact, which is put in the `adapting` state. Once the adaptation is complete, the previous state (`idle` or `executing`) is restored. In case the planner ends up in an inconsistent state, such as due to excessive delays or too many failures that prevent the achievement of the desired goals, the `solver` fact is updated to indicate a `failed` state of the planner. Further actions, such as creating a new solver with a new problem, are delegated to the rule-based system. When the execution completes all planned tasks achieving the desired goals, the state of the `solver` fact becomes `finished`. It is important to note that the `solver` fact is a crucial element in the communication and coordination between the planner and the sequencing tier, ensuring effective execution of the plan while managing exceptional cases. Figure 3 illustrates the potential states and transitions of the solvers.

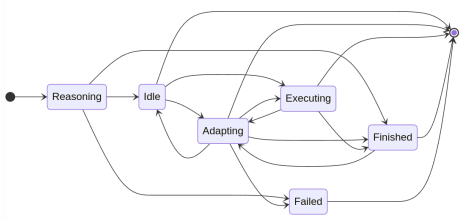


Fig. 3: State-transition diagram illustrating the potential states of the solvers.

During task execution, effective communication with the sequencing tier is crucial. Task execution in the COCO system requires confirmation from the

Function	Description
`starting(solver_id, id, type, pars, vals)`	Queries the sequencing tier to determine if the task with the given `id` in the plan managed by the `solver_id` solver, characterized by the `type` type and with parameters `pars` taking on values `vals`, can be initiated. Returns a Boolean value (by default, `TRUE`) indicating whether the task can start. If not, an optional numerical value may be provided to indicate the estimated delay for starting the activity.
`start(solver_id, id, type, pars, vals)`	Informs the sequencing tier that the task, identified by the provided parameters, has just begun. By default, this action asserts a corresponding `task` fact in the knowledge base.
`ending(solver_id, id)`	Queries the sequencing tier to determine if the task with the given `id` in the plan managed by the `solver_id` solver can be terminated. Returns a Boolean value (by default, `TRUE`) indicating whether the task can finish. If not, an optional numerical value may be provided to indicate the estimated delay for ending the activity.
`end(solver_id, id)`	Informs the sequencing tier that the task with the given `id` in the plan managed by the `solver_id` solver has just finished. By default, this action retracts the corresponding `task` fact from the knowledge base.

Table 3: Functions called during the execution of plans in the COCO system.

sequencing tier, which is aware of dynamic environmental updates. Communication is maintained by adding relevant facts to the knowledge base and invoking specific custom functions. These functions are detailed in Table 3. The `starting` function checks if a task can begin, returning a Boolean value and an optional delay estimate if initiation is not possible. The `start` function informs the sequencing tier of a task's start, updating the knowledge base. Similarly, the `ending` function assesses if a task can end, and the `end` function signals task completion, updating the knowledge base accordingly. These functions ensure clear coordination between the deliberative and sequencing levels within the COCO system.

4 The Case Study of Matera

Matera (a photo of the city is visible within Figure 2) is a beautiful city in southern Italy known for its ancient town, the "Sassi di Matera", a UNESCO World Heritage Site since 1993. In recent years, Matera has undergone a significant transformation, rebranding itself from a poverty-stricken city in the 1950s to a modern, thriving hub today, center for innovation and cultural development. The Matera 2019 program, as the European Capital of Culture, included numerous initiatives to promote the city's history, art, and cultural heritage. For the occasion, Matera has been selected to host the "House of Emerging Technologies" project, funded by the Italian Ministry of Economic Development. The project,

in particular, has implemented several technological solutions to enhance the city's urban management and improve the quality of life of citizens and visitors. One of these solutions is a data platform that collects information from sensors placed around the city, including data from citizen notifications, which can be used for analysis and decision-making. Additionally, a 3D model of the city, annotated to achieve a semantic 3D representation, provides valuable insights for urban management [20]. The project also employs Dynamic Mode Decomposition to analyze pedestrian and vehicle traffic patterns and predict future states [7]. Path planning algorithms have also been developed to help visitors optimize their visits, finding the optimal route based on user preferences such as shortest paths, minimum slopes, or maximum shade [10].

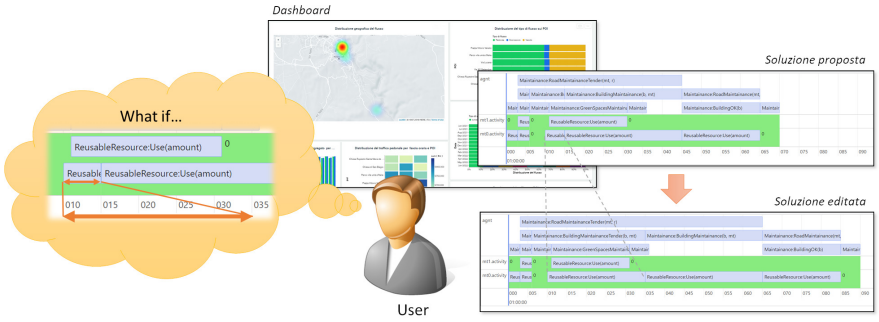


Fig. 4: The COCO web app contains a dashboard for real-time monitoring and for visualizing statistics. Plans can be visualized and edited in order to perform what-if analysis.

The COCO system[2] is one element in this ensemble, aiming to coordinate the operations of the various components, plan necessary activities, and manage part of the information received from sensors. While planning urban interventions, decision-makers and municipal technicians receive real-time suggestions on activities to undertake, and citizens receive pertinent updates on the status of the city (e.g., public events). Both decision-makers and municipal technicians can interact with COCO and, if needed, incrementally adjust the generated solutions by introducing additional constraints, thereby conducting a what-if analysis. For instance, Figure 4 displays the user-accessible web-based dashboard, presenting information about the city's state and statistics derived from sensors' history. The figure also includes a timeline representation of an executing plan, illustrating the impact of extending a task due to a user-initiated what-if analysis.

The rule-based system within COCO maintains a comprehensive dataset related to the city's state. This includes details about the managed sensor types, the specific sensors strategically positioned throughout the city, information

[2] https://github.com/ratioSolver/COCO.

about roads (e.g., known condition, length, capacity, slope, etc.), details about buildings (e.g., known condition, energy efficiency, etc.), road traffic and atmospheric simulations, predicted future states (to react to problems before they occur), a registry of users, who have system access, with their associated skills, and so on. Data generated by sensors reaches COCO via a REST API and contributes to the assertion of additional facts, which in turn can trigger the activation of rules and subsequent execution of specific actions. Similarly, the rule-based system has been expanded with user-defined functions to facilitate both direct actions (e.g., activating a sprinkler) and indirect actions (e.g., communicating with municipal technicians or decision-makers). These actions aim to influence the city and may involve suggesting decisions to decision-makers. Suppose, for example, that an air quality sensor sends a pm10 value equal to 55 µg/mc. This information results in adding a `(sensor_data aq0 04042023 55)` fact, in which `(aq0)` is the ID of the air quality sensor, 04042023 is the timestamp of the datum and 55 is the pm10 perceived amount. Since the daily limit threshold, in Italy, is 50 µg/mc, the following rule triggers the sending of a warning message to the person in charge of managing the situation.

```
(defrule pm10
    (sensor_data (id ?s_id) (timestamp ?t) (data ?pm10))
    (sensor (id ?s_id) (location ?lat ?lng) (type_id ?type))
    (sensor_type (id ?type) (name "air_quality")) (test (>= ?pm10 50))
    (user (id ?u_id)) (skill (id ?u_id) (name "air_monitoring")) =>
    (send_message ?u_id (str-cat "The air quality sensor has perceived a pm10
    value of " ?pm10 " which is above the recomended threshold of 50 µg/mc")))
```

Another scenario involves managing participatory data through a web interface where citizens can report issues concerning buildings, roads, and other public amenities. When a certain threshold of reports is reached, the rule-based system activates the deliberative tier by setting a goal for maintaining the reported asset. In modeling the problem, the monitoring of maintenance interventions is entrusted to a team of municipal technicians, who are also assigned the task of preparing the documents for the related calls for tenders, aimed at assigning the activities to the companies which, won tenders, will be awarded the contract for the effective implementation of maintenance operations. Each technician has specific skills, limiting their involvement to certain types of maintenance tasks (e.g., road and public green maintenance but not public building maintenance). Additionally, to prevent overloading, no technician can handle more than two

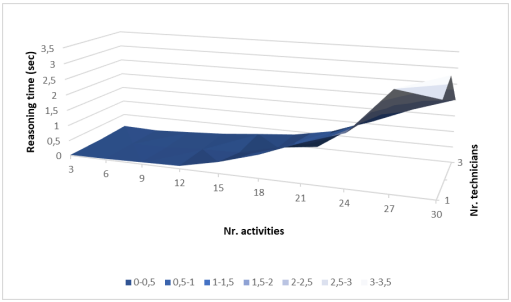


Fig. 5: Planning times based on the number of maintenance goals and the number of municipal technicians.

activities simultaneously; for instance, in case of three concurrent activities, they can be assigned to two different technicians or scheduled to overlap in a way that respects the limit of two concurrent activities per technician.

The ongoing process of defining rules encompasses both the rules-based system and the deliberative tier. Specifically, the deliberative tier addresses issues related to the maintenance of public goods, urban planning interventions, and the intricacies of procurement procedures, which must navigate the complexities of Italian bureaucracy. On the other hand, rules for the rule-based system primarily consider scenarios where predefined thresholds are exceeded. Employing machine learning techniques, particularly decision tree learning, we leverage data obtained from road simulations to formulate rules for predicting traffic jams on specific roads, based on the monitoring of a selected subset equipped with sensors. To assess the effectiveness of COCO, nonetheless, our focus turned to evaluating the efficiency of the reasoners' resolution processes on a set of benchmark problems of increasing size. This approach allowed us to estimate resolution times as the size of the addressed problems grew. Given that decision-makers might interact with COCO to perform what-if analyses during urban intervention planning, the system's response must be swift. Figure 5 illustrates the resolution times varying with the number of maintainance goals and on the number of municipal technicians in the team. Notably, despite the exponential complexity[3] of the problem, resolution times consistently remain within a few seconds, even with higher numbers of activities. It is worth to note how, the planner has greater freedom in assigning activities, resolution times decrease slightly with more technicians.

5 Conclusions

This paper introduces the COCO system, integrating a rule-based system with automated planners to support decision-making in urban management. Rule-based systems react efficiently to environmental changes, while automated planning provides a more deliberative approach to generating tasks that achieve desired goals. By combining these approaches, urban managers gain a comprehensive tool for managing various aspects of urban life. Pattern-matching techniques enhance the efficiency of rule-based systems in reactive tasks, while the deliberative component, though more computationally intensive, remains feasible for scenario generation, plan adaptation, and what-if analyses.

Defining rules is a critical and often complex process shared by both systems. In future work, we aim to explore machine learning to streamline and, where possible, automate rule definition. Additionally, changes in municipal administrations have delayed tasks like procuring and installing sensors. In the coming months, we plan to conduct experiments with real users, including municipal administrators and citizens. We have also begun applying these techniques in the cities of Catania and Milano.

[3] The planner must sequence activities while respecting temporal and resource constraints, thus solving an NP-Hard problem in this case.

References

1. Apt, K.R., Wallace, M.G.: Constraint Logic Programming Using ECL^{i}PS^e. Cambridge University Press, New York, NY, USA (2007)
2. Arnott, D., Gao, S.: Behavioral economics for decision support systems researchers. Decision Support Systems **122**, 113063 (2019). https://doi.org/https://doi.org/10.1016/j.dss.2019.05.003, https://www.sciencedirect.com/science/article/pii/S016792361930079X
3. Castelli, G., Cesta, A., Diez, M., Padula, M., Ravazzani, P., Rinaldi, G., Savazzi, S., Spagnuolo, M., Strambini, L., Tognola, G., Campana, E.F.: Urban intelligence: a modular, fully integrated, and evolving model for cities digital twinning. In: 2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT and AI (HONET-ICT). pp. 033–037 (2019). https://doi.org/10.1109/HONET.2019.8907962
4. De Benedictis, R., Beraldo, G., Cesta, A., Cortellessa, G.: Incremental timeline-based planning for efficient plan execution and adaptation. In: Dovier, A., Montanari, A., Orlandini, A. (eds.) AIxIA 2022 – Advances in Artificial Intelligence. pp. 225–240. Springer International Publishing, Cham (2023)
5. De Benedictis, R., Cesta, A.: Lifted Heuristics for Timeline-based Planning. In: ECAI-2020, 24th European Conference on Artificial Intelligence. pp. 2330–2337. Santiago de Compostela, Spain (August 2020)
6. Dechter, R.: Constraint Processing. Elsevier Morgan Kaufmann (2003)
7. Diez, M., Serani, A., Campana, E.F., Stern, F.: Data-driven modelling of ship maneuvers in waves via dynamic mode decomposition (2021)
8. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19**(1), 17–37 (1982). https://doi.org/https://doi.org/10.1016/0004-3702(82)90020-0, https://www.sciencedirect.com/science/article/pii/0004370282900200
9. Gat, E.: On Three-Layer Architectures. In: Artificial Intelligence and Mobile Robots. pp. 195–210. AAAI Press (1997)
10. Gentile, C., Stecca, G., Mancini, S., Suanno, M.: An application of the orienteering problem with time windows for scheduling visits during social events. In: Joint EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization, Bologna (Italy), June 25 - 27, 2018 (2018)
11. Gerevini, A., Serina, I.: Fast plan adaptation through planning graphs: Local and systematic search techniques. In: Chien, S.A., Kambhampati, S., Knoblock, C.A. (eds.) Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000. pp. 112–121. AAAI (2000), http://www.aaai.org/Library/AIPS/2000/aips00-012.php
12. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Publishers Inc. (2004)
13. Grosan, C., Abraham, A.: Rule-Based Expert Systems, pp. 149–185. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21004-4_7, https://doi.org/10.1007/978-3-642-21004-4_7
14. Hopgood, A.A.: Intelligent Systems for Engineers and Scientists, 3rd Edition (2016)
15. Kahneman, D.: Thinking, Fast and Slow. Farrar, Straus and Giroux, New York (2011)
16. Moreno-Monroy, A.I., Schiavina, M., Veneri, P.: Metropolitan areas in the world. delineation and population trends. Journal of Urban Economics **125**, 103242 (2021). https://doi.org/https://doi.org/10.1016/j.jue.2020.103242,

https://www.sciencedirect.com/science/article/pii/S0094119020300139, delineation of Urban Areas
17. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In: Zweben, M. and Fox, M.S. (ed.) Intelligent Scheduling, pp. 169–212. Morgan Kauffmann (1994)
18. Niu, L., Lu, J., Zhang, G.: Cognition-driven decision support for business intelligence: models, techniques, systems and applications / Li Niu, Jie Lu, and Guangquan Zhang. Springer Verlag Berlin (2009)
19. OECD, Commission, E.: Cities in the World (2020). https://doi.org/https://doi.org/https://doi.org/10.1787/d0efcbda-en, https://www.oecd-ilibrary.org/content/publication/d0efcbda-en
20. Scalas, A., Cabiddu, D., Mortara, M., Spagnuolo, M.: Potential of the geometric layer in urban digital twins. ISPRS International Journal of Geo-Information **11**(6) (2022). https://doi.org/10.3390/ijgi11060343, https://www.mdpi.com/2220-9964/11/6/343
21. Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990–2010). IEEE Transactions on Autonomous Mental Development **2**(3), 230–247 (2010). https://doi.org/10.1109/TAMD.2010.2056368
22. Tsalatsanis, A., Hozo, I., Kumar, A., Djulbegovic, B.: Dual processing model for medical decision-making: An extension to diagnostic testing. PLoS One **10**(8) (2015). https://doi.org/https://doi.org/10.1371/journal.pone.0134800