Declarative Reasoning Approaches for Agent Coordination

Filippos Gouidis, Theodore Patkos, Giorgos Flouris, and Dimitris Plexousakis

Institute of Computer Science, FO.R.T.H
Heraklion, Crete, Greece
{gouidis,patkos,fgeo,dp}@ics.forth.gr

Abstract. Reasoning about Action and Change (RAC) and Answer Set Programming (ASP) are two well-known fields in AI for logic-based reasoning. Each paradigm bears unique features and a possible integration can lead to more effective ways to address hard AI problems. In this paper, we report on implementations that embed RAC formalisms and concepts in ASP and present the experimental results obtained, building on a graph-based problem setting that introduces casual and temporal requirements.

Keywords: Answer Set Programming, Event Calculus, Reasoning about Action and Change, Multi-Agent Action Coordination

1 Introduction

Planning within a multi-agent setting is a challenging task in terms of computational capacity and representational demand, involving coordination of actions under constraints of diverse complexity. The practical impact of this kind of planning problems is evident in a multitude of domains ranging from logistics to robotics, and others [1]. Over the last decades, research in AI has contributed a repertoire of methodologies to approach multi-agent classical planning. Among them, theories for reasoning about action and change (RAC) stand out as a prominent approach. Building on the progress of classical first-order logic (FOL), formalisms such as the Situation Calculus [2] and the Event Calculus (EC) [3, 4] have been applied successfully to reasoning in dynamic domains. Nonetheless, one of the great benefits of FOL, its high expressiveness, is also one of its main limitations when confronted with the demands of practical domains. Much effort has been placed in appropriately restricting FOL, as for instance to Horn clause reasoning to allow for efficient theorem proving.

In the last decade, the progress achieved in the field of Answer Set Programming (ASP) has grown at a fast pace. ASP is a form of declarative programming oriented towards difficult, primarily NP-hard, search problems [5]. It uses alternative semantics than standard and logic programming, based on default reasoning and the notion of stable models [6], while relying on constructs that resulted in powerful and efficient systems. Moreover, recent advancements in the underlying theoretical models (e.g., [7]) and a better understanding on their basic properties managed to close significantly the conceptual gap that exists between ASP and FOL [8], offering the opportunity for the one to harness features from the other.

In this paper, we build on a setting that introduces challenging features to both fields of ASP and FOL and develop two approaches that use EC as specification language and ASP as implementation language, as well as a third approach relying exclusively on state-of-the-art ASP constructs. Consider the following example.

Example 1.1 The Intelligent Operations Center Domain: We imagine the case of a centrally-managed operations center (OC) of a smart city. The OC's dashboard receives requests from citizens that can be treated by different public services (Emergency Medical Service, Fire Department, Police Department etc). Each request requires the presence of a unit (agent) of one or more services in a certain location, in order to perform a specific activity.

The capabilities among agents may vary significantly. For example, a hospital can have both helicopters and ambulances, and these two types of agents have totally different travel times between locations in the city. Moreover, the presence of agents of one service or their operations may affect the agents of other services. For example, once a vehicle of the civil protection finishes clearing a road from rubble it will speed up the travel time of an ambulance that wants to pass this road; on the contrary, the presence of a fire fighter in a certain location may cause delays and traffic jams, hampering the speed of the ambulance.

The main responsibility of the OC is to manage the city services and decide how to effectively deploy and plan the actions of the available agents, in order to provide optimum and timely assistance to citizens. Apart from the financial gains achieved by optimizing everyday operations, the system becomes valuable during emergency response scenarios (e.g., following a major earthquake), where there is an overflow of requests and coordinating operations becomes critical. □

While non-declarative approaches have extensively been applied in problem settings similar to the one described above, extending their representational models with new dimensions is non-trivial. Furthermore,they are often prone to domain-dependent performance [9]. Our targeted objectives aim at demonstrating how expressive RAC formalisms, which are able to reason about complex phenomena, such as temporal and causal constraints, inertia, triggered events, coordination of actions, can be harmonized with the recent progress in ASP on theoretical and practical aspects. The paper reports on initial results towards comparing strong and weak points of RAC and ASP theories both from the point of view of representation and from the point of view of computational efficiency. The current and future results of this line of research aim at contributing towards a better coupling of state-of-the-art formalisms covering real-world, computationally intensive requirements. Encodings and execution guidelines are available online[1].

[1] F2LP website (last accessed 4/3/2014): `http://www.ics.forth.gr/isl/MACPDRA/`

The rest of this paper is structured as follows. After a brief introduction in Section 2 to the languages involved, we formally characterize the problem setting of Example 1.1 in Section 3. In Section 4, we elaborate on the strong and weak points of modeling this problem under different methodologies. Section 5 presents experimental results and the paper concludes with a discussion on future steps.

2 Background

2.1 Event Calculus

Action theories are logical languages for reasoning about the dynamics of changing worlds, having played a pivotal role in the development of non-monotonic logics and in formalisms to represent knowledge. The EC [3, 4] is a narrative-based many-sorted first-order language for reasoning about action and change, where the sort $\mathcal{E}$ of *events* (or actions) indicates changes in the environment, the sort $\mathcal{F}$ of *fluents* denotes time-varying properties and the sort $\mathcal{T}$ of *time-points* is used to implement a linear time structure. The calculus implements the *principle of inertia* for fluents, which captures the property that things tend to persist over time unless affected by some event, and applies the technique of *circumscription* to solve the frame problem and support default reasoning.

Different dialects have been proposed over the years; in this study, we axiomatize our domains based on the Discrete-time Event Calculus (DEC) [10] and the recently proposed Functional Event Calculus (FEC) [11]. DEC defines a set of predicates to express which fluents hold when ($HoldsAt \subseteq \mathcal{F} \times \mathcal{T}$), which events happen ($Happens \subseteq \mathcal{E} \times \mathcal{T}$), which their effects are ($Initiates$, $Terminates$, $Releases \subseteq \mathcal{E} \times \mathcal{F} \times \mathcal{T}$) and whether a fluent is subject to the law of inertia or released from it ($ReleasedAt \subseteq \mathcal{F} \times \mathcal{T}$).

FEC, on the other hand, generalizes the EC to include non-binary (i.e. non-truth-valued) fluents taking values from the sort $\mathcal{V}$. In accordance to DEC, the key predicates and functions are $Happens \subseteq \mathcal{E} \times \mathcal{T}$, $ValueOf : \mathcal{F} \times \mathcal{T} \rightarrow \mathcal{V}$, $CausesValue \subseteq \mathcal{E} \times \mathcal{F} \times \mathcal{V} \times \mathcal{T}$, $PossVal \subseteq \mathcal{F} \times \mathcal{V}$. Non-determinism and triggered actions are supported by both formalisms. Along with the set of domain-independent rules that axiomatize the notions of inertia, causality and effect, the execution of reasoning tasks is performed with a set of domain-dependent axioms expressing the dynamics of the world.

2.2 Answer Set Programming

Answer set programming (ASP) is a recently developed programming technique combining declarativeness, modularity and expressiveness. It is founded on logic programming answer sets semantics, which drive the computation of stable models (answer sets). The procedure followed on most of ASP solvers is an enhanced version of DPLL algorithm. ASP is gaining increasing popularity due to its ability to combine an expressive, non-complex language over powerful solvers.

An Answer Set Program is a set of rules of the form:

Rule: A_0 :- L_1 , . . . , L_{k-1},not L_k, ...,not L_n .

where L_j are atoms and *not* represents negation-as-failure. The set of literals $\{L_1, \ldots, L_n\}$ are called the body of the rule and A_0 the head. Intuitively, the head of a rule has to be true whenever all its body literals are true in the following sense: a non negated literal L_i is true if it has a derivation and a negated one, `not` L_i, is true if the atom L_i does not have one. According to stable model semantics, only atoms appearing in some head can appear in answer sets. Furthermore, derivations have to be acyclic, a feature that is important to model reachability. Rules with an empty body are called facts and their head is unconditionally true, i.e., it appears in all answer sets. Rules with an empty head are called integrity constraints and are used to reject answer set candidates.

ASP has already proven its potential in expressiveness in comparison to other declarative approaches, enabling the representation of phenomena for commonsense and nonmonotonic reasoning ([12, 13]), while its solvers outperform satisfiability-based and constraint-programming solvers in many domains [14, 15]. The success of ASP is demonstrated in a wide variety of fields that spans from hardware design and phylogenetic inference to the Semantic Web. In this study, we use the most recent ASP implementation developed by Potsdam Answer Set Solving Collection (Potassco), named Clingo[2]. It combines the grounder Gringo and solver Clasp and encompasses many utilities, such as detailed tuning of grounding and solving, utilization of useful built-in functions and also the ability to integrate scripts written in Lua and Python languages, through which high flexibility to the developer is possible.

3 Smart City Operations Center: Formal Definition

We assume a set $\mathcal{AG} = \{\alpha_1, \alpha_2, \ldots\}$ of *agents* having different *types*, denoted by $\mathcal{TP} = \{\tau_1, \tau_2, \ldots\}$. Each type represents different capabilities (such as cars or helicopters). Agents also belong to different *services* $\mathcal{SV} = \{\sigma_1, \sigma_2, \ldots\}$, representing agencies in the smart city, such as the Police or the Fire Brigade. We denote by τ^α and σ^α the type and service, respectively, that agent α belongs to.

The smart city contains a set $\mathcal{LC} = \{\lambda_1, \lambda_2, \ldots\}$ of *locations.* Each location is associated with a (possibly empty) set of services, representing the fact that this location requires agents from said services to visit it; this set of services is denoted by $Srv^{\lambda_i} \subseteq \mathcal{SV}$ for some location λ_i.

The conceptual model for the operations center system is a state-transition system described by the 4-tuple $\mathcal{OC} = \langle \mathcal{S}, \mathcal{AC}, \mathcal{EV}, \gamma \rangle$, where:

- $\mathcal{S} = \{s_1, s_2, \ldots\}$ is a finite set of states that capture the assignment of agents and services to specific locations and also the connections between locations with specific costs[3];

[2] Clingo website (last accessed 4/3/2014): http://potassco.sourceforge.net/

[3] Due to space limitations, we leave implicit certain aspects, such as the propositions that characterize each state; these are formally defined in subsequent sections.

- $\mathcal{AC} = \{move\}$ is the set of possible actions, with $move : \mathcal{AG} \times \mathcal{LC} \rightarrow \mathcal{LC}$ being the only action that an agent can perform, i.e., move between locations;
- $\mathcal{EV} = \{changeCost\}$ the set of possible events, with $changeCost : \mathcal{TP} \times \mathcal{LC} \times \mathcal{TP} \times \mathcal{LC} \times \mathcal{LC} \rightarrow \mathbb{Z}^+$ a function expressing that when some agent type is at a specific location, it changes the cost of the connection between two locations for a given type of agent. Such events may be triggered by the change in location in each state;
- $\gamma : \mathcal{S} \times 2^{\mathcal{AC}} \rightarrow 2^{\mathcal{S}}$ is a state transition function. Multiple actions may occur concurrently, but we implicitly assume only one action for each agent.

Note that we model no specific agent action for the servicing of tasks at a location. In the future, we will also include servicing actions with durations and order (priorities) among them; for the time being, the arrival of an agent at a location causes also the successful execution of the task required. The system $\mathcal{OC}$ is fully observable (i.e., the initial state is fully known), deterministic and no exogenous actions or events can occur (i.e., the only changes to its state are due to agent actions or events triggered by these actions).

As explained above, the $\mathcal{OC}$ will issue a set of orders to the agents, essentially creating a path that each agent should follow (i.e., a sequence of locations). Given $\mathcal{OC} = \langle \mathcal{S}, \mathcal{AC}, \mathcal{EV}, \gamma \rangle$, an initial state s_0 and a set of goal states $S_g \subset S$, a plan π is defined as usual, as a sequence of nonempty sets of actions $(\{c_1^1, ...\}, ..., \{c_1^k, ...\})$ corresponding to a sequence of state transitions $(s_0, s_1 ..., s_k)$, such that $\gamma(s_0, \{c_1^1, ...\}) = s_1$, $\gamma(s_1, \{c_1^2, ...\}) = s_2$, ..., $\gamma(s_{k-1}, \{c_1^k, ...\}) = s_k$ and $s_k \in S_g$. In our case, a plan π_{acc} is called *acceptable* iff for all locations λ_j in the smart city, and for all $\sigma \in Srv^{\lambda_j}$, there exists an agent α and a state s_i $(0 \geq i \geq k)$ produced by π_{acc}, such that $\sigma^\alpha = \sigma$ and the agent is at location λ_j at state s_i.

The objective of the $\mathcal{OC}$ is to develop a plan that will service all nodes in the least possible time. Let $T^\pi(i, \alpha)$ denote the time required for agent α to execute the i^{th} $move(\alpha, \lambda_j)$ action of its own plan, i.e., the time it spent at λ_j plus the time required to travel from λ_j to the destination dictated by the action. The *service time* for a plan π, denoted by $\widehat{\pi}$, is defined as $\widehat{\pi} = max_\alpha\{\sum_i T^\pi(i, \alpha)\}$.Now, a plan π_{opt} is called *optimal* iff it is acceptable and there is no acceptable plan π'_{acc} such that $\widehat{\pi'}_{acc} < \widehat{\pi}_{opt}$. We denote as $\mathcal{SCOC}$ the problem of finding an optimal plan for a given $\mathcal{OC}$ instance of a smart city.

Example 2.1 Simple Action Coordination: Consider the graph presented on Fig. 1a. In the initial state agent α_1 is located at λ_1 and needs to reach λ_4, while agent α_2 located at λ_2 needs to reach λ_5. Moreover, the arrival of α_2 at λ_2 reduces the cost of traveling from λ_1 to λ_2 from 9 to 3 time units, i.e., it increases the speed of agents traveling through this edge by a factor of 3.

Fig. 1b presents the situation that would occur if all agents acted towards minimizing the time required to execute their own plan: α_2 would go directly to λ_5 resulting in a total overall execution time of 11 time units. Given our definition of an optimum plan, the objective of agents is to minimize the overall servicing time. According to this scheme, α_2 first visits λ_3, affecting the travel time of α_1, as shown in Fig. 1c. Notice that when α_2 arrives at λ_3, agent α_1

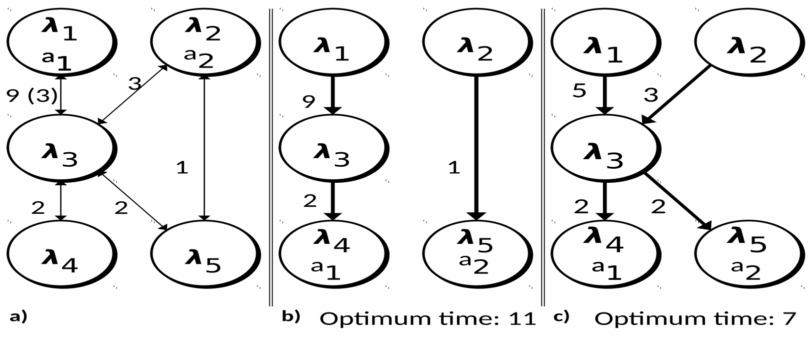

Fig. 1: Agent α_1 must reach λ_4 and α_2 λ_5. The presence of α_2 at λ_3 causes the cost of traveling from λ_1 to λ_3 to decrease from 9 to 3 time units.

already covered one third of the distance to λ_3. As such, this plan yields an overall servicing time of 7 time units and is the optimal one. □

4 Representation and Reasoning

The general structure of the $\mathcal{SCOC}$ setting requires planning with a combination of features, such as temporal and causal constraints. The problem incorporates characteristics of simpler frameworks, such as variations of the Traveling Salesman Problem, distance graphs and temporal reasoning with precedence ordering. Furthermore, it extends them in various ways, as for instance with dynamically changing edge costs. While the individual problems are extensively studied in relevant literature, their interplay in a unified framework still remains an open problem which attracts the interest of research community, mainly due to their impact in many real-world domains [9]. Such a setting requires expressive formalisms, in order both to support the complex phenomena that emerge and to allow for a formal verification of their properties. In this section, we describe how the EC and ASP can be applied to approach representational and practical issues related to the problem of $\mathcal{SCOC}$, revealing strong and weak points of each.

4.1 Representing $\mathcal{SCOC}$ with Event Calculus Axiomatizations

The $\mathcal{SCOC}$ planning problem formulates a demanding domain; while RAC theories are well suited to express some of its aspects, others prove to be more challenging, as we discuss next. We chose the EC as the specification language to describe the domain, not only due to its ability to model a multitude of commonsense phenomena, but also because of the availability of tools that can support our reasoning tasks.

Both FEC and DEC have been used to model the setting; we concentrate in this subsection on FEC, whose ability to model functional fluents offers greater flexibility, even though both theories are comparable for the given domain. We picture the smart city as a graph, whose edges have weights that may change dynamically as a result of occurring events. To simplify the presentation we assume one service and one agent type; the axiomatization can trivially be extended to the more general case, by simply using a different graph per agent type/service. In compliance with the notation introduced in the previous sections, let $ag, ag_1, ...$ denote variables of the $\mathcal{AG}$ sort, $l, l_1, ...$ variables of the $\mathcal{E}$ sort, while variables $num, num_1, ...$ denote positive reals[4].

The following domain closure axioms define all fluents and actions needed to model $\mathcal{SCOC}$, in order to reason about the state of agents (i.e., being at a location or moving), the state of locations (i.e., served or not) and the distance traveled:

$$f = At(ag) \lor f = Moving(ag) \lor f = RemDist(ag, l_1, l_2) \lor f = Step(l_1, l_2) \lor f = Served(l).$$
$$e = Departs(ag, l_1, l_2) \lor e = Arrives(ag, l) \lor UpdateRemDist(ag, l_1, l_2, num).$$

Fluent *Step* denotes how much distance the agent can cover in one timepoint along the edge (l_1, l_2), while *RemDist* captures the distance that remains to be traveled. *Step* is subject to change, as it depends on the state of the world. We further assume uniqueness of names axioms for actions and fluents. The possible values for these fluents are appropriated restricted:

$$PossVal(At(ag), l).\ PossVal(Step(l_1, l_2), num).\ PossVal(RemDist(ag, l_1, l_2), num).$$
$$PossVal(Moving(ag), v) \equiv v = True \lor v = False.$$
$$PossVal(Served(l), v) \equiv v = True \lor v = False.$$

Certain predicates are also defined. For instance, $Connected(l_1, l_2, num)$ denotes edges and the corresponding distance between locations.

In comparison to other action theories, the explicit representation of time inside EC predicates facilitates the developer in expressing complex temporal expressions, necessary in $\mathcal{SCOC}$ to model for instance *actions with durations*, such as traveling for a given amount of time. Moreover, the calculus has established solutions to the frame, ramification and qualification problems, relieving the developer from the tedious work of explicitly writing frame or minimization axioms. Many aspects of the domain can easily be described by axioms expressing *context-dependent effects of actions*, *action preconditions* and *state constraints*, with existentially quantified variables whenever needed:

$$CausesValue(Arrives(ag, l), at(ag), l, t) \leftarrow ValueOf(Moving(ag), t) = True.$$
$$Happens(Departs(ag, l_1, l_2), t) \rightarrow ValueOf(At(ag), t) = l_1.$$
$$ValueOf(At(ag), t) = l_1 \land ValueOf(At(ag), t) = l_2 \rightarrow l_1 = l_2.$$
$$Happens(Departs(ag, l_1, l_2), t) \rightarrow \exists num Connected(l_1, l_2, num).$$

In order to handle metric distances between locations or calculate traveled distances, $\mathcal{SCOC}$ calls for extensive use of *numerical operations* to be incorporated in the domain description. For instance, we have to recalculate the *Step* fluent every time certain agent actions affect it, such as when some agent arrives

[4] All variables in formulae are implicitly universally quantified, unless stated otherwise. Moreover, we assume $\mathcal{E} \supseteq \mathcal{AC} \cup \mathcal{EV}$, i.e., events axiomatized by the Event Calculus refer both to agent actions and triggered events.

at, serves or leaves a particular location. We use the predicate $AffectsCost(ag, l, l_1, l_2, num, v)$ to model different variations of the *costChange* event introduced in Section 3. For instance, when $v = 1$ (resp. $v = 2$, $v = 3$) $AffectsCost$ denotes that the cost of traveling from l_1 to l_2 becomes num when agent ag arrives at (resp. is present at, departs from) location l. The following axiom models the case when $v = 3$ (the rest are similarly defined):

$$CausesValue(Departs(ag, l), Step(l_1, l_2), (num_1/num), t) \leftarrow$$
$$[Connected(l_1, l_2, num_1) \wedge ValueOf(At(ag), t) = l \wedge affectsSpeed(ag, l, l_1, l_2, num, 3)].$$

Despite the simplicity of such axioms, the introduction of numerical variables leads to an explosion of grounded terms having tremendous impact in performance, as discussed in Section 5, calling for special measures to be adopted.

Finally, *triggered events*, i.e., events that occur when the world is in a particular state, are a significant leverage in modeling real-world domains [16]. In our case, we use triggered events to keep track of the distance that an agents needs to travel before reaching the destination location:

$$Happens(UpdateRemDist(ag, l_1, l_2, (num_1 - num_2)), t) \leftarrow ValueOf(Moving(ag), t) = True \wedge$$
$$ValueOf(RemDist(ag, l_1, l_2), t) = num_1 \wedge ValueOf(Step(l_1, l_2), t - 1) = num_2.$$

Note that, in contrast to most benchmark problems in action theories, in our case the duration of certain actions, such as the agent's travel is not known beforehand, rather it needs to be calculated on-the-fly. Our modeling adopts the simple solution of recalculating the remaining distance at every timepoint according to the distance the agent has traveled in the previous timepoint (notice that *Step* refers to $t - 1$ in the previous axiom). A variation of this problem with static edge weights has also been implemented to show the difference in performance when action duration is known *a priori*. ASP constructs can provide radical solutions, as we argue in the next section.

Finally, the axiomatization needs also to include the description of the initial state, specifying the location and state of all agents, as well as the goal state, i.e., $ValueOf(Served(n), T_{opt}) = True$ for some timepoint T_{opt}. Note that since the problem we solve is a planning problem, we do not specify completion of the $Happens$ predicate, letting the reasoner produce all combinations of event occurrences that can lead to the satisfaction of the goal state. Of course, T_{opt} is not known beforehand.

4.2 Representing $\mathcal{SCOC}$ in Answer Set Programming

This subsection describes an alternative modeling that uses ASP as specification language for describing $\mathcal{SCOC}$, aimed at exploiting the potential of state-of-the-art ASP solvers. Given the structure of the problem, we based our representation on standard methods found in literature for encoding graph traversal, as given for instance in [17]. Due to the dynamic nature of the problem, we extended the methodology with a treatment of time. In this way, atoms representing dynamic attributes contain a variable accounting for time. As before, we simplify the setting and assume only one agency and every node of the graph has to be visited at least by one of the agents.

To accommodate planning, Clingo offers a special functionality where reasoning progresses incrementally. Specifically, a special variable, which denotes timepoints in our case, is acting as a place-holder that increases by a constant step number to perform grounding and solving in consecutive steps, until an answer set satisfying the goal state is found. To accomplish this, the program is divided into 3 independent parts: the basic, the cumulative and the volatile part. The former contains the definitions that are used throughout the execution, as well as the initial state, while the latter specifies the goal condition. The cumulative part, on the other hand, incorporates all rules and constraints that have to be grounded every time the reasoning progresses by one step.

The state of the graph is specified by predicates, such as *in(Ag1,Nd1,1)* and *edge(Nd1,Nd2,W,1)* contained in the basic part. In order to represent the displacement of the agents, rules able to express cardinality constraints are used:

$$0\{on\_the\_road(AG,X,Y,C,t) : edge(X,Y,C,t)\}1 : -agent(AG), in(AG,X,t).$$

This rule states that an agent located at a node at a certain time-point can initiate *at most* one movement and this movement should have as destination a node that is connected (i.e., *edge*) to the node that is currently located at. Such rules give significant leverage to the developer, offering a compact way to introduce various types of restrictions.

In order to avoid the overload of grounded terms introduced in the Event Calculus encodings when numerical operations are in place, we embedded in our ASP encodings *external predicates*, a special functionality offered by Clingo. The truth value of these predicates can be decided by scripts written in the Lua language, without requiring grounding or disturbing execution during reasoning. Such an external predicate is @*new_cost* appearing in the following rule:

$$on\_the\_road(AG,X,Y,C\_NEW,t) : -on\_the\_road(AG,X,Y,C\_OLD,t-1), 0 < C\_OLD,$$
$$e(X,Y,W\_OLD,t-1), e(X,Y,W\_NEW,t), C\_NEW = @new\_cost(W\_NEW,W\_OLD,C\_OLD,t).$$

The rule calculates the remaining distance for agents on the move. The performance gains obtained when executing such computationally demanding tasks in parallel with reasoning is depicted in the evaluation discussed in Section 5.

While the built-in constructs described before offer enhanced functionalities to support declarative reasoning, certain aspects of $\mathcal{SCOC}$ were not handled as conveniently as with the EC encodings. The representation of inertia, which had to be explicitly defined in all time-dependent rules, and the treatment of time in general, are characteristic examples:

$$-edge(X,Y,C2,t) : -edge(X,Y,C,t), edge(X,Y,C2,t-1), C2! = C.$$
$$edge(X,Y,C2,t) : -edge(X,Y,C2,t-1), not\ -edge(X,Y,C2,t).$$

These rules model the weight of edges at each timepoint, having the law of inertia explicitly expressed. The first rule implies that if for two consecutive time moments an edge has different weights, the earlier value must become obsolete the next time moment, while the second rule indicates that if an edge's value has not become obsolete, it is conserved for the next time moment ("−" denotes strict negation). Similar behavior has to be designed for other atoms, whose value may change over time. The ease of accommodating such phenomena with the EC and their direct application to new features with minumum effort, often referred to as elaboration tolerance, becomes easily evident.

Finally, the constraint expressing the goal condition, i.e., whether all nodes have been visited, is expressed as follows and added in the volatile part:

$$:-not\ reached(X,t), node(X), query(t).$$

We additionally made use of Clingo's built-in function *minimize*, in order to achieve a second-level of optimization:

$$\#minimize\{C : on\_the\_road(AG,X,Y,C,t)\}$$

With this expression we can define a secondary criterion to classify optimal plans, when more than one are found. Specifically, we choose the one with minimum distances traveled by all agents, denoted by variable C. Special treatment of such expressions allows the Clingo solver to calculate solutions effectively.

5 Experimental Evaluation

5.1 Preparation

Recent progress in generalizing the definition of stable model semantics used in ASP [7] has opened the way for highly expressive formalisms to be reformulated in ASP encodings and take advantage of the several efficient implementations that are available. Specifically, the precise characterization of the correspondence between stable models and circumscription used by many theories for reasoning about action and change, such as the Situation and the EC, has permitted the reformulation of the latter in ASP. For that purpose, we used the F2LP[5] tool to transform our circumscriptive DEC-based axiomatization into a logic program that can be executed with ASP reasoners. This is important since not all first-order formulae can be transformed into the clausal form used in ASP solvers while preserving stable models. F2LP applies the translation developed in [18] that guarantees that the ASP encoding created as output is equivalent to the circumscription-based axiomatization.

While the FEC-based axiomatization can also become input to F2LP, we preferred to use the dedicated reasoner and encoding style for FEC theories developed in [19]. The importance of this tool relies on its capacity to support reasoning with very expressive classes of problems with minimum effort on the developer's side. Specifically, it can execute the epistemic extension of FEC [11], which we plan to integrate in future variations of the $\mathcal{SCOC}$ setting, when for instance not all agent locations will be known initially.

Due to the fact that FEC and DEC are implemented in a non-incremental way, we created a non-incremental version of the ASP implementation (ASP-non), in addition to the incremental one (ASP-inc), to study the differences and allow for a more direct comparison. In particular, we first executed each problem instance with ASP-inc in order to find optimal times and then we used those times to ran with the rest of the programs.

[5] F2LP website (last accessed 4/3/2014): `http://reasoning.eas.asu.edu/f2lp/`

Our results are given in terms of overall time for solution (analyzed further in grounding and solving time) and atoms produced. All experiments were conducted on a workstation having 2 Intel eight-core CPU of 2.30 GHZ and 384 GB of RAM memory. For the computation of the answer sets version 4.2.1 of Clingo was used. A time limit of 15 minutes held for each test; if a solution was not found within the time limit the result was considered unknown. Also, all implementations were tuned to generate only one answer set. All encodings, results and a script for executing the encodings are available online.

5.2 Results

We designed 2 sets of experiments, one for a simple problem variation where the topology of the graphs remained unchanged during planning (s-$\mathcal{SCOC}$) and another where edge costs could changed dynamically according to agents' locations (d-$\mathcal{SCOC}$). Tables 1 present relevant statistics for both variations. Each experiment considered graphs of size 5, 10 and 15 nodes with an increasing number of agents scattered randomly in each case. A set of rules also assigned random values to edges (ranging from 1 to 10), and also dictated how the location of agents could affect those values in the d-$\mathcal{SCOC}$case.

At first glance, it is evident that d-$\mathcal{SCOC}$is a much harder problem than s-$\mathcal{SCOC}$, as reflected on the times required to find optimal plans. We also noticed that for more complex problem instances, the incremental approach to problem solving is generally more expensive than the non-incremental one, requiring the reasoner to interchange between grounding and solving while converging to a solution. As already mentioned, the incremental approach is the only way of finding a plan when the number of necessary steps is unknown. Therefore, one important conclusion is to incorporate in future executions the functionality offered by Clingo to manually set lower and upper bounds for initiating and terminating the incremental reasoning process; approximate values for these bounds can easily be determined given the initial state.

It is interesting to notice also that increasing the number of agents available to serve tasks manages to reduce significantly plan finding times, even though this is not always reflected in the number of atoms produced. This is related to the length of the produced plans, which is shorter when more agents are available, despite the fact that more alternative routes need to be considered. In fact, what we found considering bigger, as well as less complex graphs than cliques, is that this performance gain converges to a number of agents specific for each topology, after which no significant profit is measurable.

Noticeable also is the difference in performance between the EC and the pure ASP implementations. This difference is primarily attributed to the cost of handling the numerical manipulations by the former. Specifically, in order to model edge traversal, we considered as the smallest distance that an agent can travel in one timepoint to be a a tenth of the unit distance. That is, if the maximum cost of any edge in a graph is 8, the smallest distance covered in one step is 0.1, requiring number variables to range between 80 possible values. For the ASP case though, we relied on external predicates to model and calculate

Table 1: Experimental Results for the s-$\mathcal{SCOC}$and d-$\mathcal{SCOC}$case. (times are in seconds).

Graph Size			5			10			15		
# of agents			1	3	5	1	6	10	1	8	15
s-$\mathcal{SCOC}$	ASP-inc	$T^{\dagger}$	0.035	0	0	1.82	0.01	0.01	46.98	0.01	0.01
		$G^{\diamond}$	0.01	0	0	1.80	0	0	46.96	0	0
		$S^{\triangle}$	0.025	0	0	0.02	0.01	0.01	0.02	0.01	0.01
		A°	894	539	401	2691	1989	2310	4462	2856	2889
	ASP-non	$T^{\dagger}$	0.02	0	0	7.27	0.01	0.02	85.35	0.06	0.05
		$G^{\diamond}$	0.01	0	0	7.25	0	0	85.33	0.01	0.01
		$S^{\triangle}$	0.01	0	0	0	0.01	0.02	0.02	0.05	0.04
		A°	1717	1029	742	6019	4473	5241	11205	6805	6730
	FEC	$T^{\dagger}$	0.34	0.03	0.02	8.70	0.17	0.20	431.44	0.06	0.52
		$G^{\diamond}$	0.11	0	0	8.02	0	0	429.28	0.01	0.01
		$S^{\triangle}$	0.23	0.03	0.02	0.68	0.17	0.20	2.16	0.05	0.51
		A°	10533	4847	4224	31578	14965	18241	58184	6805	34812
	DEC	$T^{\dagger}$	0.05	0.02	0.02	0.68	0.22	0.47	21.76	0.99	2.46
		$G^{\diamond}$	0.01	0	0	0.59	0.04	0.19	21.45	0.23	1.06
		$S^{\triangle}$	0.04	0.02	0.02	0.09	0.18	0.28	0.31	0.76	1.40
		A°	3526	4049	4497	14559	22663	30368	33566	51821	74032
d-$\mathcal{SCOC}$	ASP-inc	$T^{\dagger}$	0.03	0.01	0.01	0.71	0.02	0.03	473.43	0.20	0.10
		$G^{\diamond}$	0.01	0	0	0.61	0	0	473.08	0	0
		$S^{\triangle}$	0.02	0.01	0.01	0.10	0.02	0.03	0.35	0.20	0.10
		A°	1179	1629	2359	10966	3363	6260	43174	4355	8094
	ASP-non	$T^{\dagger}$	0.01	0.01	0.01	0.07	0.03	0.01	90.46	0.06	0.08
		$G^{\diamond}$	0	0	0	0.02	0	0	89.72	0	0.01
		$S^{\triangle}$	0.01	0.01	0.01	0.05	0.03	0.01	0.74	0.06	0.07
		A°	1120	1490	2282	7530	3020	5799	34862	5732	7360
	FEC	$T^{\dagger}$	230.31	135.92	132.50	N/A	284.80	460.90	N/A	408.86	N/A
		$G^{\diamond}$	0	0	0	N/A	0.07	0.13	N/A	0.07	N/A
		$S^{\triangle}$	230.31	135.92	132.50	N/A	284.73	460.77	N/A	408.79	N/A
		A°	406064	347891	327901	N/A	677987	948587	N/A	1080183	N/A
	DEC	$T^{\dagger}$	20.04	27.83	29.30	N/A	86.65	146.59	N/A	116.60	228.99
		$G^{\diamond}$	0.01	0.14	0.03	N/A	0.11	0.50	N/A	0.45	2.22
		$S^{\triangle}$	20.03	27.69	29.27	N/A	86.54	146.09	N/A	116.15	226.77
		A°	60868	89823	114522	N/A	277090	445634	N/A	528443	947717

† Total Time $^{\diamond}$ Grounding Time $^{\triangle}$ Solving Time $^{\circ}$ Atoms Produced

such values, significantly reducing the number of grounded atoms. As depicted in the tables, these are orders of magnitude fewer in the ASP encodings with respect to the EC encodings. This conclusion will play key role in our future implementations that will be driven by the goal of seamlessly combining the structures offered by state-of-the-art ASP tools into EC encodings.

Finally, we notice that in the majority of cases the dominant factor affecting performance is the time spent by the grounder to instantiate the problem, whereas solving time is usually negligible. We note here that our encodings did not emphasize on performance and the solutions adopted leave space for further improvements. We expect that the incorporation of heuristics along with modelings more tailored to the specific execution style of Gringo will achieve better results. Nevertheless, we wish to stress that the recently released reactive ASP solver oClingo is expected to have enormous impact in problems such as the one studied here. oClingo is able to perform online reasoning without having to re-initialize the solving process every time new information arrives. This facility will be harnessed to provide replanning capabilities for future variations of $\mathcal{SCOC}$ that will additionally allow for unpredictable occurrences of external events to change the state of the system.

6 Discussion and Conclusion

In this paper, we contrasted contemporary and prominent approaches of the fields of RAC and ASP, both from the representational and the practical standpoint. We initiate a line of research that focuses on those aspects where a synergetic application can prove more fruitful, building on recent theoretical and applied advancements in both fields.

Closest to ours is the work conducted in the context of the housekeeping robotics domain [20, 21]. Research there investigates collaborative planning by considering ASP and the high-level action description language $\mathcal{C}+$ for domain specification, leaning towards the former as more appropriate to formalize the concepts involved. Yet, temporal relations, which are inherent in this kind of settings, are much more conveniently handled with the EC than with $\mathcal{C}+$ or domain-specific solutions. This type of problem can be addressed by other techniques, such as graphplan or integer linear programming. But, as $\mathcal{SCOC}$ will expand to include more features in the future, with non-deterministic events (stochastic and unpredictable), durational, prioritized tasks, and the capability of re-planning being the most imminent ones, the broadness of the tools and theories we presented in this study will be the first candidates to resolve them. For this reason, we also plan to consider other EC dialects, such as the Cached Event Calculus [22], and study their performance.

Furthermore, the problem bears strong similarities to the general construct of the Traveling Salesperson Problem and its many variations. The progress that has been and still is being made in analyzing their theoretical properties and developing efficient procedural algorithms, as for instance in [9], may provide inspiration for developing more efficient, real-world implementations for $\mathcal{SCOC}$. As we showed, the use of external predicates and the strong coupling of declarative reasoning with scripting languages may achieve significant performance gains.

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
2. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
3. Kowalski, R., Sergot, M.: A Logic-based Calculus of Events. New Generation Computing **4**(1) (1986) 67–95
4. Miller, R., Shanahan, M.: Some alternative formulations of the event calculus. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, London, UK, Springer-Verlag (2002) 452–490

5. Lifschitz, V.: What is answer set programming? In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI. (2008) 1594–1597
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: International Logic Programming Conference and Symposium. (1988) 1070–1080
7. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175**(1) (2011) 236–263
8. Denecker, M., Vennekens, J., Vlaeminck, H., Wittocx, J., Bruynooghe, M.: Answer set programming's contributions to classical logic. In: Logic programming, knowledge representation, and nonmonotonic reasoning. Volume 6565. (2011) 12–32
9. Kumar, T.S., Cirillo, M., Koenig, S.: On the traveling salesman problem with simple temporal constraints. In: 10th Symposium of Abstraction, Reformulation, and Approximation(SARA'13). (2013) 73–79
10. Mueller, E.: Commonsense Reasoning. 1st edn. Morgan Kaufmann (2006)
11. Miller, R., Morgenstern, L., Patkos, T.: Reasoning about knowledge and action in an epistemic event calculus. In: 11th International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'13). (2013)
12. Eiter, T., Ianni, G., Krennwallner, T.: Reasoning web. semantic technologies for information systems. (2009) 40–110
13. Coban, E., Erdem, E., Ture, F.: Comparing ASP, CP, ILP on two Challenging Applications: Wire Routing and Haplotype Inference. In: Proc. of the 2nd International Workshop on Logic and Search (LaSh 2008). (2008)
14. Kim, T.W., Lee, J., Palla, R.: Circumscriptive event calculus as answer set programming. In: 21st International Joint Conference on Artificial Intelligence (IJCAI-09). (2009) 823–829
15. Celik, M., Erdogan, H., Tahaoglu, F., Uras, T., Erdem, E.: Comparing ASP and CP on four grid puzzles. In: Proc. of the 16th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'09). (2009)
16. Tran, N., Baral, C.: Reasoning about Triggered Actions in AnsProlog and Its Application to Molecular Interactions in Cells. In: 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR2004). (2004) 554–564
17. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2012)
18. Lee, J., Palla, R.: Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. Journal of Artificial Intelligence Research **43**(1) (2012) 571–620
19. Ma, J., Miller, R., Morgenstern, L., Patkos, T.: An epistemic event calculus for asp-based reasoning about knowledge of the past, present and future. In: Proc. of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19). (2013)
20. Erdem, E., Aker, E., Patoglu, V.: Answer set programming for collaborative housekeeping robotics: Representation, reasoning, and execution. Intelligent Service Robotics **5**(4) (November 2012) 275–291
21. Aker, E., Erdogan, A., Erdem, E., Patoglu, V.: Causal reasoning for planning and coordination of multiple housekeeping robots. In: Logic Programming and Nonmonotonic Reasoning. (2011) 311–316
22. Chittaro, L., Montanari, A.: Efficient temporal reasoning in the cached event calculus. Computational Intelligence **12**(3) (1996) 359–382