Automated Software Engineering (2019) 26:513–557
https://doi.org/10.1007/s10515-019-00255-5

Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing

Davide Dell'Anna[1] · Fabiano Dalpiaz[1] · Mehdi Dastani[1]

Received: 4 November 2018 / Accepted: 8 April 2019 / Published online: 22 April 2019

Abstract
Sociotechnical systems (STSs) are defined by the interaction between technical systems, like software and machines, and social entities, like humans and organizations. The entities within an STS are autonomous, thus weakly controllable, and the environment where the STS operates is highly dynamic. As a result, the design artifacts that represent the requirements of an STS, such as requirements models, may end up being invalid when the system operates, for the autonomous entities do not comply with the requirements, or the environment changes. In this paper, we present a framework that uses runtime execution data to support the runtime validation of requirements models and to guide the evolution of an STS. We propose two types of evolution: (i) *manual*: the analyst uses Bayesian inference to discover which assumptions in a requirements model are invalid and manually adjusts the system or its model; and (ii) *automated*: requirements are iteratively revised by an hill climbing algorithm searching for requirements that maximize the achievement of the stakeholders' objectives. We evaluate the effectiveness of different revision heuristics on a smart traffic simulation applied to an exemplar from the self-adaptive systems literature. The results show that our heuristics, informed by runtime execution data, outperform standard uninformed heuristics, in terms of convergence speed, solution quality, and stability. Moreover, the algorithms show good resilience to noise introduced into the execution data.

Keywords Requirements engineering · Sociotechnical systems · Bayesian networks · Requirements revision

✉ Davide Dell'Anna
d.dellanna@uu.nl

Fabiano Dalpiaz
f.dalpiaz@uu.nl

Mehdi Dastani
m.m.dastani@uu.nl

1 Utrecht University, Utrecht, The Netherlands

1 Introduction

For over forty years, researchers and practitioners in software and requirements engineering (RE) have proposed and experimented methods and tools to specify and evolve the requirements of *software systems* (Lehman and Ramil 2003; van Lamsweerde et al. 1998). However, the increasing embedding of cyber-physical and sociotechnical systems (STSs) (Sommerville et al. 2012; Dalpiaz et al. 2013; Chopra et al. 2014) in our lives poses new challenges for the RE discipline.

A smart city, for example, is an STS that includes heterogeneous entities such as pedestrians, drivers, vehicles, bicycles, traffic lights and signs, speed cameras, and road regulations. This STS is governed by the city council that can alter the road regulations and control artifacts such as traffic lights to best achieve the system objectives (e.g., to reduce jams). However, many entities (humans and vehicles) are autonomous and therefore weakly controllable (Chopra et al. 2014).

The autonomy of the participating entities and the dynamic, open nature of STSs (Dalpiaz et al. 2013; Sommerville et al. 2012) entail that anticipating all the possible states of the system and transitions between them is not an option (Whittle et al. 2010; Lehman 2005) and the compliance of the system with its requirements cannot be guaranteed. Runtime requirements monitoring and diagnosis are therefore essential activities to determine system compliance with its requirements, which may eventually trigger evolution or adaptation mechanisms.

Several frameworks (Wang et al. 2009; Robinson 2006; Fickas and Feather 1995) have been proposed to support runtime requirement monitoring and diagnosis. Many of such approaches represent requirements via requirements models (Bencomo et al. 2010; Wang et al. 2009), and analyze system execution data in terms of requirements satisfaction.

The self-adaptive systems literature goes beyond diagnosis, and proposes solutions to adapt a system when their requirements are threatened (Krupitzer et al. 2015; Bencomo et al. 2013; De Lemos et al. 2013). Self-adaptive systems search for a new system configuration that is expected to outperform the current one in achieving the system requirements.

Unfortunately, state-of-the-art approaches implicitly rely on the correctness of the requirements model. When designing a system, however, requirements engineers make *assumptions* about requirements, their satisfaction conditions, and the environment in which the requirements should be satisfied (Lehman and Ramil 2003; Lehman 2005; Boness et al. 2008, 2011; Ali et al. 2011). This is even more true for STSs, due to the autonomy of the participating entities and the volatility of the environment.

In this paper, we propose a framework (see Fig. 1) for the adaptation and evolution of STSs that challenges the validity of the assumptions in a requirements model. We use Bayesian Networks to learn the relationship between the satisfaction of requirements and overall system objectives. Based on such information, the framework can be used to (i) validate the assumptions in the model and let the analyst manually evolve the system or its model; and (ii) automatically revise the requirements model by determining the most appropriate requirements for the achievement of the overall system objectives.

Specifically, we make the following contributions to the literature:

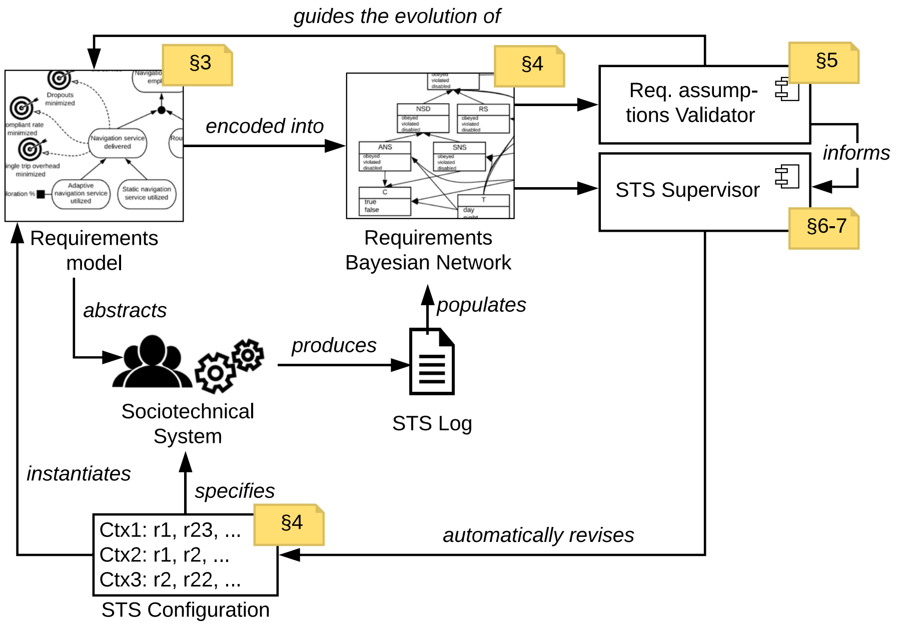


Fig. 1 Overview of the framework for STS evolution presented in the paper

- We propose *Requirement Bayesian Network*s (*RBN*) as the runtime counterpart of the requirements models created at design-time; an *RBN* is populated with execution logs and apprehends the causal relationships between requirements and overall system objectives in the different operating contexts;
- We explain how a human analyst can validate the design-time assumptions in a requirements model through the use of an *RBN*;
- We present an automated requirements revision mechanism that can be used for the system to identify sets of requirements that maximize the achievement of the system objectives in each operating context. A first version of our heuristics was presented in Dell'Anna et al. (2018b), with a focus on identifying optimal norms to govern the behaviour of a multiagent system. The approach employs a variant of the hill climbing optimization technique to iteratively revise the norms based on their effectiveness in achieving the overall objectives of the system. In this paper, we extend this work along three dimensions: (i) we apply our heuristics to the case of hierarchical requirements models, as opposed to flat norm sets; (ii) we formally define the concepts of requirement variant, system configuration, and requirement revision; and (iii) we present a substantial evaluation of our algorithms;
- Via a smart traffic simulation applied to a mid-sized city, we evaluate how effective our revision mechanisms are at finding good-enough requirements.

Organization Section 2 presents our research background. Section 3 introduces the smart traffic working example that we use throughout the paper. Section 4 defines *RBNs* and shows how to map requirements models to them. Section 5 presents different types of design-time assumptions and describes how to validate them based on

RBN information. Section 6 elaborates on our framework for automatic requirements revision. Section 7 reports on our evaluation using a smart traffic simulator. Section 8 reports on related work. Section 9 discusses our work and sketches future work.

2 Background

We present the key background for this paper: (i) requirements models; and (ii) Bayesian Networks for representing and learning knowledge.

2.1 Requirements models

Requirements models have been used and studied extensively in RE. As pointed out by the IREB handbook of requirements modeling (Cziharz et al. 2016), such models can be created with different purposes, including specifying a system, supporting testing, and increasing clarity. Depending on the purpose, the analysts may decide to represent the information structure, scenarios, goals and objectives, or other aspects of the system under development and its environment.

Here, we focus on hierarchical requirements models, that organize the requirements for a system as refinement trees, where high-level objectives—explaining the *raison d'être* for the requirements (Yu and Mylopoulos 1998)—are specified in terms of more specific requirements and system functions. In particular, we take inspiration from the rich literature on goal models (Van Lamsweerde 2009; Yu and Mylopoulos 1998; Dalpiaz et al. 2016), but choose a general notation that does not commit to a specific modeling language.

A small requirements model for a car wash service is shown in Fig. 2. We distinguish between *requirements* and *objectives*. Requirements (rounded rectangles) define the behavior that the designer expects the entities within the STS to perform. For example, having *cars cleaned*, or doing so via a *fully automated wash*. Objectives (targets with an arrow) express the conditions that denote stakeholder satisfaction with the system; for example, *Customer retention rate over 80% per year* indicates that the car wash owners do not simply want cars to be cleaned, but they aim at retaining most customers to sustain their business.

We organize requirements in hierarchies via AND- and XOR-*refinements*. For an AND-refined requirement to be satisfied, all of its sub-requirements need to be satisfied. For example, in order to have cars cleaned, both the interior and the exterior of cars should be cleaned, and positive opinions should be reported by the drivers. A XOR-refined requirement describes possible *mutually exclusive* ways for its achievement. For example, exterior cleaning can be done either via a fully automated wash or through a manual wash.

The expected impact of requirements on objectives is represented via *aims at* links, which denote positive contributions from requirements to objectives (Giorgini et al. 2002). In Fig. 2, the designer expects that washing the car interior will support achieving an 80% customer retention rate.

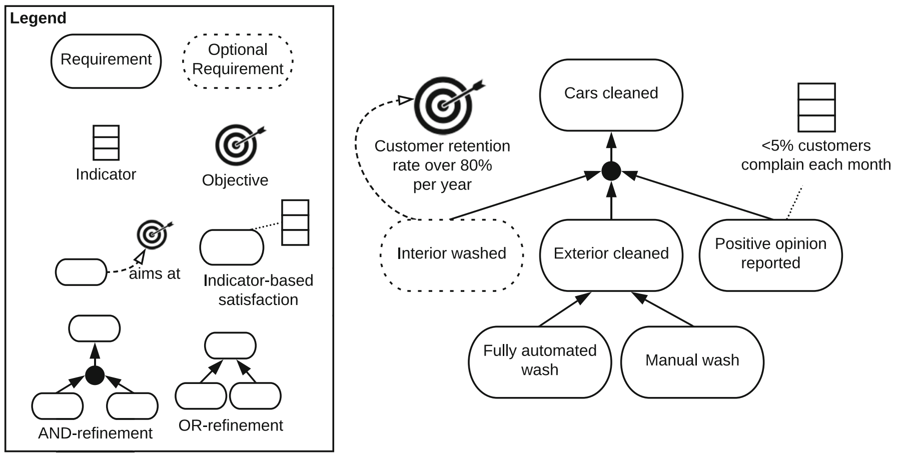


Fig. 2 A small requirements model for a car wash service

We use *indicators* to qualify the satisfaction of requirements. Requirements that are not associated with an indicator (called regular requirements) are required to be satisfied by every instance of the said requirement. For example, *manual wash* is satisfied when all cars starting a manual wash are actually washed. Conversely, the satisfaction of requirements that are associated with an indicator (called aggregate requirements) is determined by aggregating a number of instances of that requirement. For example, *positive opinion reported* is satisfied when less than 5% of the customers complain. The analyst should specify the frequency for evaluating the indicator (e.g., monthly).

Finally, requirements can be optional (dotted border), indicating that they can either be selected or not selected. For example, *interior washed* is not necessary for having *cars cleaned*.

We formalize our requirements model in Definition 1, which is used in Sect. 4 to explain how requirements models are mapped to Bayesian Networks.

Definition 1 (*Requirements model*) A requirements model is a tuple $\mathcal{RM} = \langle(\mathcal{R}, ch, d), \mathcal{O}, \mathcal{SC}, cl, type, sc, opt\rangle$, where

- $(\mathcal{R}, ch, d)$ is an AND-OR tree, where $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a set of requirements, $ch : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ is a function that returns the children of a requirement, and $d : \mathcal{R} \rightarrow \{\texttt{AND}, \texttt{XOR}\}$ is partial function that determines the type of refinement of a requirement with children;
- $\mathcal{O} = \{O_1, \ldots, O_m\}$ is a set of objectives;
- $\mathcal{SC} = \{SC_1, \ldots, SC_{n+m}\}$ is a set of satisfaction conditions for requirements, objectives, and indicators (see for instance Table 2);
- $cl : \mathcal{R} \rightarrow 2^{\mathcal{O}}$ is a function that maps requirements to the objectives that they aim at;
- $type : \mathcal{R} \rightarrow \{\texttt{agg}, \texttt{reg}\}$ is a function determining whether a requirement is *aggregate* or *regular* (all instances should be achieved);

- $sc : \mathcal{R} \cup \mathcal{O} \rightarrow \mathcal{SC}$ is a function that determines the satisfaction condition of requirements and objectives; and
- $opt : \mathcal{R} \rightarrow \{\texttt{true}, \texttt{false}\}$ is a function determining whether or not a requirement is optional.

The requirements model in Fig. 2 can therefore be expressed according to Definition 1. A partial formalization is the following:

- $\mathcal{R} = \{$*cars cleaned*, ..., *positive opinion reported*$\}$
- $ch($*cars cleaned*$) = \{$*interior washed*, *exterior cleaned*, *positive opinion reported*$\}$, $ch($*exterior cleaned*$) = \{$*fully automated wash*, *manual wash*$\}$
- $d($*cars cleaned*$) = \texttt{AND}$, $d($*exterior cleaned*$) = \texttt{XOR}$
- $\mathcal{O} = \{$*customer retention rate over 80% per year*$\}$
- $cl($*interior washed*$) = \{$*customer retention rate over 80% per year*$\}$
- $opt($*interior washed*$) = \texttt{true}$, ...

2.2 Bayesian networks

Bayesian Networks have been widely used in many fields, ranging from medicine to forensics, as knowledge representation structures for learning and reasoning about the inter-dependencies between their nodes (Russell and Norvig 2010).

In software engineering, their applications include evaluating software reliability (Doguc and Ramirez-Marquez 2009), estimating software effort (Mendes and Mosley 2008), modeling software quality (Misirli and Bener 2014), and defect prediction (Fenton et al. 2007). In RE, Bayesian Networks have been employed both for the runtime verification of requirements (Filieri et al. 2012) and for decision making (Bencomo et al. 2013).

Definition 2 (*Bayesian network*) A Bayesian network (Russell and Norvig 2010) $\mathcal{B} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ is a directed acyclic graph, where:

- $\mathcal{X}$ is the set of all nodes, each corresponding to a random variable in probability theory with a discrete or continuous domain (i.e., the set of possible values the node can take).
- $\mathcal{A}$ is the set of directed links (arrows) connecting pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y. The set of parents of a node Y is denoted as ***Parents****(Y)*.
- $\mathcal{P}$ is a set of $|\mathcal{X}|$ conditional probability distributions. Each node $X \in \mathcal{X}$ is associated with a conditional probability distribution $\mathbf{P}(X|$***Parents****(X)*$)$ that quantifies the effect of the parents on the node.

Note that in the context of Bayesian Networks we use the notation shown in Table 1. The pair $(\mathcal{X}, \mathcal{A})$ is called the *structure* of the Bayesian Network $(\mathcal{X}, \mathcal{A}, \mathcal{P})$. An evidence $\mathbf{e}$ is a revealed (observed) assignment of values for some or all of the random variables in the Bayesian Network, i.e., $\mathbf{e} = \{X_v | X \in \mathbf{X}\}$ with $\mathbf{X} \subseteq \mathcal{X}$ and v a possible value in the domain of the variables.

Given the set $\mathcal{X}$ of all the nodes in a Bayesian Network $\mathcal{B}$ and a (possibly empty) evidence $\mathbf{e}$, reasoning with $\mathcal{B}$ generally means to determine the distribution $\mathbf{P}(\mathbf{X}|\mathbf{e})$,

Table 1 A summary of the notation used for Bayesian networks

Notation	Description
$X, Y, \ldots$	Random variables (italic uppercase)
$\mathbf{X}, \mathbf{Y}, \ldots$	Set of random variables (bold uppercase)
$v_1, v_2, \ldots$	Value in the domain of a random variable (italic lowercase)
$\mathbf{x}, \mathbf{y}, \ldots$	Assignment of values to a set of nodes (bold lowercase)
X_v	$(X = v)$, assignment of value v to a random variable X
$\mathbf{X}_v$	Assignment of value v to all nodes in $\mathbf{X} \subseteq \mathcal{X}$
X_{act}	$\neg X_{dis} = \neg(X = disabled)$, the fact: X is not *disabled*
$\mathbf{P}$	Probability distribution
P	Single probability

with $\mathbf{X} \subseteq \mathcal{X}$ a set of nodes of which we want to discover the probability distribution (e.g., $\mathbf{P}(X|Y_v)$ is the probability distribution of the values of the random variable X, given that value v is observed for variable Y).

3 The CrowdNavExt smart traffic simulator

In this paper, we study the evolution of an STS through computer simulation, a powerful tool for testing alternative configurations prior to changing the real environment, which is particularly adequate to analyze the behavior of autonomous agents in a large-scale real setting (Luke et al. 2005; Tsvetovat and Carley 2004; Wu et al. 2015).

We start from the CrowdNav *smart traffic simulator*, an exemplar from the self-adaptive systems literature (Schmid et al. 2017) that simulates traffic scenarios in the middle-sized city of Eichstädt, in Germany, with 450 streets and 1200 intersections. We propose CrowdNavExt,[1] which introduces multiple types of navigation services as well as different ways of managing junctions, in line with the requirements model of Fig. 3, described in the following.

The city council of Eichstädt aims at improving the traffic by achieving two overall objectives: ensuring an *average trip overhead below 250%* compared to the theoretical traveling time without traffic, and guaranteeing *less than 4 accidents per day*. To achieve such objectives, the city council plans to opportunely manage junctions in the city and to offer to the cars a Centralized Navigation Service (CNS) in addition to the cars' personal navigation system. Due to the highly dynamic nature of the city, drivers and vehicles can behave differenlty in different contexts. In this paper we consider two *contextual properties* (*Time* and *Weather*), that can assume two values each: *Time* can either take *day* or *night*, while *Weather* can either be *normal* or *extreme*.

Two top-level requirements, set by the city council to achieve the objectives, are the following: *at least 10% of the cars in the city shall always use the offered CNS* (the requirement *NS* in Fig. 3 and the associated indicator), and *every junction in the city is opportunely managed* (requirement *J*).

[1] CrowdNavExt's code repository: https://bitbucket.org/dellannadavide/crowdnavext.

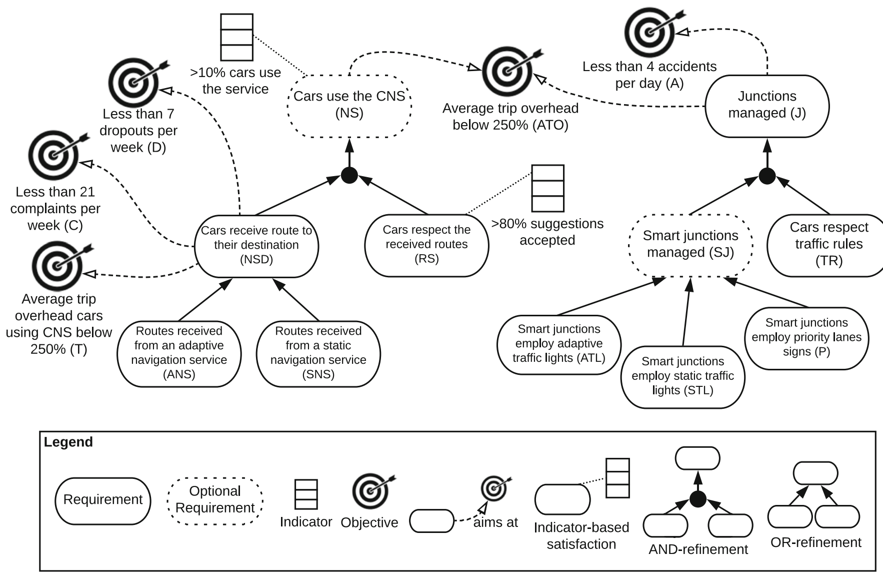

>10% cars use the service
Less than 7 dropouts per week (D)
Cars use the CNS (NS)
Average trip overhead below 250% (ATO)
Less than 4 accidents per day (A)
Junctions managed (J)
Less than 21 complaints per week (C)
Cars receive route to their destination (NSD)
Cars respect the received routes (RS)
>80% suggestions accepted
Smart junctions managed (SJ)
Cars respect traffic rules (TR)
Average trip overhead cars using CNS below 250% (T)
Routes received from an adaptive navigation service (ANS)
Routes received from a static navigation service (SNS)
Smart junctions employ adaptive traffic lights (ATL)
Smart junctions employ static traffic lights (STL)
Smart junctions employ priority lanes signs (P)
Legend
Requirement
Optional Requirement
Indicator
Objective
aims at
Indicator-based satisfaction
AND-refinement
OR-refinement

Fig. 3 A requirements model for the smart traffic simulation

To satisfy the requirement *NS*, two sub-requirements are *assumed* to be necessary: *whenever a car starts a trip toward a destination, the car shall receive a route from the Central Navigation Service* (*NSD*) and *at least 80% of all the route suggestions given by the CNS are respected by the cars equipped with the CNS* (*RS*). *NSD* can be met by either employing a self-adaptive navigation service (*ANS*) (Schmid et al. 2017) or a static navigation service (*SNS*). In our simulator, each car relies on a navigation service to determine its route from origin to destination: 90% of the vehicles use their personal navigation service (the default routing algorithm of the simulator), while the remaining 10% are smart cars that can use a centralized navigation service. When smart cars do not use the centralized navigation service, they use their own navigator as normal cars.

The *NSD* requirement is assumed to help achieve three additional objectives concerning the satisfaction of the users of the navigation service: *less than 21 complaints per week* (*C*); *less than 7 dropouts per week* (*D*), i.e., cars that decide to stop using the CNS; and *average trip overhead cars using CNS below 250%* (*T*), for some cars using the CNS will be suggested paths to explore in order for the CNS to identify optimal paths.

To satisfy the requirement *J*, two sub-requirements are assumed to be necessary: *every junction that is equipped with smart panels (called smart junctions) shall display on the panel the prescribed traffic rule* (*SJ*), and *every car shall respect the traffic rules prescribed by the junctions in the city* (*TR*). *SJ* can be met by either displaying on the panels traffic lights that adapt their timing according to the traffic (*ATL*), or by displaying regular traffic lights (*STL*), or by displaying which of the lanes in the junctions has priority (*P*). When no management is prescribed for smart junctions, the vehicles approaching the junctions follow the default priority-to-the-right rule.

Table 2 Satisfaction conditions of the requirements and objectives in our scenario

Obj/req	Satisfied	Evaluated every
NS	$>10\%$ vehicles in the city is using the CNS	Time instant
NSD	Every time a CNS-equipped car starts a trip, it receives a route from the CNS	Trip
ANS	Every time a CNS-equipped car starts a trip, it receives a route from an ANS	Trip
SNS	Every time a CNS-equipped car starts a trip, it receives a route from a SNS	Trip
RS	$>80\%$ of all CNS suggestions has been accepted	Week
J	All sub-requirements are satisfied	Time instant
SJ	Every smart junction displays the traffic rules on its panel	Time instant
ATL	Every smart junction displays adaptive traffic lights	Time instant
STL	Every smart junction displays regular traffic lights	Time instant
P	Every smart junction displays priority lanes signs	Time instant
TR	Every time a car crosses a junction, it satisfies the displayed traffic rule	Car at junction
ATO	The average trip overhead of all the vehicles has been below 250%	Week
A	The number of accidents is below 28	Week
C	The number of complaints received is below 30	Week
D	The number of dropouts is below 7	Week
T	The average trip overhead of vehicles using the CNS has been below 250%	Week

Table 2 describes precisely the conditions for requirement monitors to determine requirements and objectives satisfaction.

4 From requirements models to Bayesian networks

In this section, we define the type of Bayesian Network (called *Requirement Bayesian Network*, or *RBN*) that we use for supporting requirements evolution, and we explain how to automatically generate the structure of an *RBN* from requirements models as presented in Definition 1.

4.1 Requirement Bayesian Network

Let $\mathcal{CP} = \{\mathcal{CP}_i, \ldots, \mathcal{CP}_k\}$ be a set of monitorable contextual properties of the STS system (i.e., monitorable environmental variables that determine the operating context of the system, e.g., *Time*, *Weather*), each associated with a domain of values (e.g., *Weather* can be either *normal* or *extreme*).

Definition 3 (*Requirement Bayesian Network*) A *Requirement Bayesian Network* $\mathcal{RBN} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ is a Bayesian Network where:

- $\mathcal{X} = \mathbf{R} \cup \mathbf{O} \cup \mathbf{C}$ is a set of nodes, representing random variables in probability theory. The sets **R**, **O** and **C** are disjoint.

  - $\mathbf{R}$ consists of *requirement nodes*. Each node $R \in \mathbf{R}$ corresponds to a requirement and has a discrete domain of 3 possible values: *obeyed*, *violated* and *disabled*.
  - $\mathbf{O}$ consists of *objective nodes*. Each node $O \in \mathbf{O}$ corresponds to a boolean objective and has a discrete domain of 2 values: *true* and *false*.
  - $\mathbf{C}$ consists of *context nodes*. Each node $C \in \mathbf{C}$ corresponds to a contextual property $\mathcal{CP}_i \in \mathcal{CP}$ and can have discrete or continuous domain.
- $\mathcal{A} \subseteq (\mathbf{R} \times \mathbf{R}) \cup (\mathbf{C} \times \mathbf{R}) \cup (\mathbf{C} \times \mathbf{O}) \cup (\mathbf{R} \times \mathbf{O})$ is the set of arrows connecting pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y.
- $\mathcal{P}$ is a set of conditional probability distributions, each one associated with a node in $\mathcal{X}$ and quantifying the effect of the parents on the node.

An evidence $\mathbf{c}$ for all the context nodes $\mathbf{C}$ is an observation for a certain context (e.g., *Time* has value *day* and *Weather* has value *normal*). For simplicity, we call *context* also the associated evidence in the RBN.

Note that when we refer to nodes of a specific type, unless otherwise specified, we use the corresponding notation convention, e.g., R refers to a node in $\mathbf{R}$, $\mathbf{c}$ refers to an assignment of values of nodes in $\mathbf{C}$, $\mathbf{R}_{viol}$ refers to an assignment of value *violated* to a set of requirement nodes $\mathbf{R}$, etc.

4.2 From requirements models to requirement Bayesian networks

We introduce the function *RM2BNS* that generates the *structure* of an $\mathcal{RBN}$ (Definition 3) from a requirements model $\mathcal{RM}$ (Definition 1). *RM2BNS* maps a requirements model $\mathcal{RM}$ and a set of contextual properties $\mathcal{CP}$ to an $\mathcal{RBN}$ structure $(\mathcal{X}, \mathcal{A})$. Note that the probability distributions $\mathcal{P}$ of an $\mathcal{RBN}$ do not depend on the source requirements model, but are populated from the system's execution log at run-time. Therefore, $\mathcal{P}$ will not be considered in this section.

As a preliminary notion, we denote the set of requirements contributing to (aiming at) an objective O in $\mathcal{RM}$ as $cont\_desc(O)$. A requirement R contributes to an objective O if R is a descendant of O and it is either a leaf requirement, or it is of type *aggregate* but has no ancestor of type *aggregate*:

$$\begin{aligned} &cont\_desc(O) \\ &\quad = (desc(O) \setminus \{R' | R' \in desc(R), R \in desc(O), type(R) = \texttt{agg}\}) \setminus \\ &\quad \{R | ch(R) \neq \emptyset, type(R) \neq \texttt{agg}\} \end{aligned} \tag{1}$$

where $desc(R)$ is the set the descendant of R (similar for O). For instance, in the requirements model of Fig. 3, $cont\_desc(D) = \{ANS, SNS\}$.

Definition 4 (*RM2BNS*) Given a requirements model $\mathcal{RM} = \langle(\mathcal{R}, ch, d), \mathcal{O}, \mathcal{SC}, cl, type, sc, opt\rangle$ and a set of contextual properties $\mathcal{CP}$, the function *RM2BNS* returns the structure of a *Requirement Bayesian Network*; formally, $RM2BNS(\mathcal{RM}, \mathcal{CP}) = (\mathcal{X}, \mathcal{A})$, where

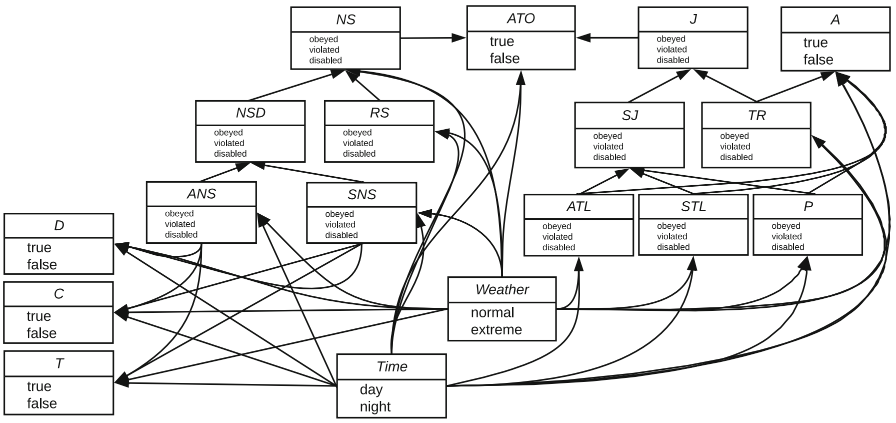


Fig. 4 The $\mathcal{RBN}$ structure defined by *RM2BNS* applied to the requirements model of Fig. 3

- $\mathcal{X} = \mathcal{R} \cup \mathcal{O} \cup \mathcal{CP}$
- $\mathcal{A} = \{(R, O) \mid R \in cont\_desc(O)\} \cup$
$\{(R_1, R_2) \mid R_1 \in ch(R_2)\} \cup$
$\{(C, R) \mid C \in \mathcal{CP}, R \in \mathcal{R}, (type(R) = \texttt{agg} \vee ch(R) = \emptyset)\} \cup$
$\{(C, O) \mid C \in \mathcal{CP}, O \in \mathcal{O}\}$

Intuitively, $\mathcal{A}$ contains (i) an arrow from a requirement node R to an objective O if R is a contributing descendant of O (see function *cont_desc* above), (ii) an arrow from a sub-requirement R_1 to its parent requirement R_2, (iii) an arrow from a context node C to each requirement node R that represents either a leaf requirement ($ch(R) = \emptyset$) or a requirement with an indicator, and (iv) an arrow from each context node C to each objective O.

Figure 4 reports the structure of the $\mathcal{RBN}$ that is generated by applying Definition 4 to the requirements model of Fig. 3.

Besides reflecting the requirements model's topology, the network also introduces the context variables. The resulting structure of the *Requirement Bayesian Network*, which consists of requirement, objective and context nodes, allows to analyze the assumptions in different operating contexts (see Sect. 5). In an $\mathcal{RBN}$, every context node is parent of all the objective nodes. This indicates that the achievement of objectives is not only due to the satisfaction (or presence) of requirements, but also to events that occur in the environment. Context nodes are also parents of all the requirement nodes whose satisfaction is not exclusively determined by their hierarchical structure in the AND/OR tree, but can also be affected by the context in which they are applied.

We choose a three-values discrete domain for the requirement nodes to make the network more versatile: while the *obeyed* and *violated* values allow to evaluate assumptions about the satisfaction or violation of requirements (e.g., the *requirement satisfiability assumption* in Sect. 5), the *disabled* value supports XOR-refined requirements. To update the conditional probability distribution of a node, it is necessary to

Table 3 Part of the dataset used to train the BN of Fig. 4 and obtained from monitoring the execution of the system in Sect. 3

Weather	*Time*	*NS*	*NSD*	...	*TR*	*ATO*	*A*	*C*	*D*	*T*
norm	night	viol	ob	...	ob	T	T	T	T	F
norm	day	ob	ob	...	ob	F	F	T	F	T
norm	day	ob	ob	...	ob	F	F	T	F	F
extr	night	viol	ob	...	ob	T	T	T	T	F
extr	day	ob	dis	...	ob	T	F	T	T	F
extr	day	ob	ob	...	ob	T	F	T	T	T
...										

Columns *ANS*, *SNS*, *RS*, *J*, *SJ*, *ATL*, *STL* and *P* are omitted due to space reasons

provide evidence for both the node and all of its parents. In case of a XOR-refined requirement, we obtain evidence only for one of the parents (sub-requirements) at a time. The *disabled* value allows therefore to perform the update also in such case.

4.3 Populating the *RBN*: data collection

Table 3 reports a sample dataset that can be obtained from monitoring the requirement and objective satisfaction from the log for the working example of Sect. 3. The values that each of the variables assumes belongs to its domain as specified in Sect. 4.1 (e.g., *obeyed*, *violated*, *disabled* for requirement nodes, *true* or *false* for objective nodes). Such dataset can be used to train the *RBN* of Fig. 4 and learn the set of conditional probability distributions $\mathcal{P}$.

A discussion of learning techniques (e.g., classical Bayesian learning) is out of the scope of this paper; we refer the interested reader to the existing literature (Russell and Norvig 2010; Spiegelhalter et al. 1993). Also, we do not analyze requirements monitoring mechanisms [e.g., EEAT (Robinson 2006)]. In the following, we assume to have a trained *RBN*.

5 Design-time assumptions and their validation

Requirements models contain *assumptions* that might be, or become, invalid in practice (Ali et al. 2011; Lehman and Ramil 2003; Lehman 2005). In this section, we describe six types of assumptions made by the designer of a system during the definition of a requirements model (as per Definition 1), and we propose a mechanism to determine the validity of such assumptions by using an *RBN* trained with system execution data. As shown in Fig. 1, this information can be used by the analysts to guide the evolution of an STS.

We introduce the notion of *degree of validity* (δ in the following) for an assumption as a real number in the range $[-1, +1]$. $\delta = +1$ denotes a fully valid assumption, $\delta = -1$ indicates a fully incorrect assumption, and the intermediate values describe an assumption with partial validity.

δ is computed as a difference between two probabilities, representing the collected positive and negative evidence for the validity of that assumption, respectively. Thus, if the collected positive evidence is close to 1 and the negative evidence is close to 0, δ will be close to +1. Values around 0 show that the assumption is only partly valid since the positive and negative evidences for the validity have similar strength.

5.1 Types of design-time assumptions

We take as a baseline the types of assumptions by Ali et al. (2011) and extend the list to support the structure of our requirements models. Note that the assumptions defined below are made *implicitly* by defining the structure of a requirements model. Therefore, even though they can be associated with a certain element of the requirements model (e.g., with an arrow or a node of the model), they are not explicitly represented in the model [unlike, e.g., the work by Boness et al. 2011].

Requirement satisfiability assumption The hypothesis that in a specific operating context, a requirement is satisfied (e.g., *in context day-extreme, the requirement RS is satisfied*). Figure 3 contains 11 requirement satisfiability assumptions (each one associated with a requirement) for each of the four possible operating contexts.

Given a context $\mathbf{c}$ and a requirement node R, the degree of validity of the associated requirement satisfiability assumption in context $\mathbf{c}$ is

$$\delta_S(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{c}) - P(R_{viol} \mid \mathbf{c}) \tag{2}$$

Objective achievement assumption The hypothesis that in a specific operating context, an objective is achieved (e.g., *in context day-extreme, the objective ATO is achieved*). Figure 3 contains 5 objective achievement assumptions (each one associated with an objective) for each of the four possible operating contexts.

Given a context $\mathbf{c}$ and an objective node O, the degree of validity of the associated objective achievement assumption in context $\mathbf{c}$ is

$$\delta_O(O, \mathbf{c}) = P(O_{true} \mid \mathbf{c}) - P(O_{false} \mid \mathbf{c}) \tag{3}$$

Contribution assumption The hypothesis that in a specific operating context, there is a positive synergy between the satisfaction of a requirement and the achievement of an objective connected via an *aims at* link (e.g., *in context day-extreme, there is a positive synergy between the satisfaction of requirement NS and the achievement of the objective ATO*). Figure 3 contains 6 contribution assumptions (each one associated with an *aims at* link) for each operating context.

Given a context $\mathbf{c}$, a requirement node R and an objective node O, the degree of validity of a contribution assumption is:

$$\delta_C(O, R, \mathbf{c}) = P(O_{true} \mid R_{ob} \wedge \mathbf{c}) - P(O_{true} \mid R_{viol} \wedge \mathbf{c}) \tag{4}$$

Notice that the degree of validity of *negative* contribution assumptions, if considered in the requirements model (omitted in this paper), due to the boolean nature of the objective nodes, can be calculated as $-\delta_C$.

Refinement assumption The hypothesis that in a specific operating context, the satisfaction of an AND-refined requirement depends on the satisfaction of all its sub-requirements (e.g.,*in context day-extreme, to satisfy the requirement NS both the requirements NSD and RS shall be satisfied*), and a XOR-refined requirement is satisfied only when one and only one of its sub-requirements is satisfied (e.g.,*in context day-extreme, the requirement NSD is satisfied when either ANS or SNS are satisfied*). Figure 3 contains 2 AND-refinement assumptions and 2 XOR-refinement assumptions (each one associated with a refinement) for each of the 4 operating contexts.

Given a context $\mathbf{c}$, a requirement node R and the set $\mathbf{R'} \in \mathbf{R}$ of its requirement nodes parents, let $\mathbf{r}$ be the disjunction of all possible assignments of values to variables in $\mathbf{R'}$ excluding the assignment $\mathbf{R'}_{ob}$, let $\mathbf{r1ob}$ be the disjunction of all possible assignments of values to variables in $\mathbf{R'}$ such that only one variable takes value *obeyed*, and let $\mathbf{ro}$ be the disjunction of all possible assignments of values to variables in $\mathbf{R'}$ excluding the assignments in $\mathbf{r1ob}$.

$$\delta_{AND}(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{R'}_{ob} \wedge \mathbf{c}) - P(R_{ob} \mid \mathbf{r} \wedge \mathbf{c}) \tag{5}$$

$$\delta_{XOR}(R, \mathbf{c}) = P(R_{ob} \mid \mathbf{r1ob} \wedge \mathbf{c}) - P(R_{ob} \mid \mathbf{ro} \wedge \mathbf{c}) \tag{6}$$

For example, the degree of validity of the AND-refinement assumption of the requirement *NS* in Fig. 3 unfolds as follows:

$$\begin{aligned}\delta_{AND}(NS, \mathbf{c}) = P(NS_{ob} \mid NSD_{ob} \wedge RS_{ob} \wedge \mathbf{c}) \\ - P(NS_{ob} \mid \neg(NSD_{ob} \wedge RS_{ob}) \wedge \mathbf{c})\end{aligned} \tag{7}$$

Adoptability assumption The hypothesis that in a specific operating context, there is a positive synergy between the satisfaction of a requirement and the satisfaction of each one of its sub-requirements separately (e.g., *in context day-extreme, there is a positive synergy between the satisfaction of the requirement SNS and the satisfaction of the requirement NSD*). Figure 3 contains 9 adoptability assumptions (each one associated with a link between a sub-requirement and a requirement) for each operating context.

Notice that, while refinement assumptions concern *one-to-many* relationships (i.e., between one requirement and all of its children), adoptability assumptions concern *one-to-one* relationships (i.e., between a requirement and each of its sub-requirements separately).

Given a context $\mathbf{c}$ and two requirement nodes R and R' such that R' is parent of R, the degree of validity of the associated adoptability assumption in context $\mathbf{c}$ is

$$\delta_{AD}(R, R', \mathbf{c}) = P(R_{ob} \mid R'_{ob} \wedge \mathbf{c}) - P(R_{ob} \mid R'_{viol} \wedge \mathbf{c}) \tag{8}$$

Requirement necessity assumption The hypothesis that in a specific operating context, the activation of a specific requirement is necessary condition for achieving all the objectives (e.g., *in context day-extreme, to achieve the five objectives ATO, A, C, D, T together, the requirement ANS must be activated*). Figure 3 contains 11 requirement necessity assumptions (each one associated with a requirement) for each operating context.

This assumption concerns the *activation* of a requirement, regardless of its satisfaction; i.e., it is the hypothesis that, in order to achieve the objectives, it is better to keep active a requirement rather than disabling it.

Given a context **c**, a requirement node R and a set of objective nodes **O**, the degree of validity of the associated requirement necessity assumption in context **c** is

$$\delta_N(R, \mathbf{O}, \mathbf{c}) = P(\mathbf{O}_{true} \mid R_{act} \wedge \mathbf{c}) - P(\mathbf{O}_{true} \mid R_{dis} \wedge \mathbf{c}) \tag{9}$$

5.2 Validating assumptions

The requirements model of Fig. 3, despite its simplicity, contains 184 assumptions (46 for each operating context, as described above) that the requirements engineer who constructed it has implicitly made. This calls for automated mechanisms that assist requirements engineers in validating such many assumptions.

Table 4 reports an evaluation of the validity of the assumptions for the requirements model of Fig. 3 when executing the simulator of Sect. 3. Specifically, we ran the simulator in all the operating contexts and we collected from the simulation logs a dataset of about 4.6 millions rows, part of which is reported in Table 3. We created an *RBN* for our scenario using the mapping function *RM2BNS*; this led to the network shown in Fig. 4. Then, we trained such network using the dataset obtained from the smart traffic simulation. For the learning, we relied on the functionality offered by the bnlearn R package (Scutari 2009). At this stage, we could evaluate the assumptions.

The calculated degree of validity of the assumptions underlying the requirements model can be used by the designer of the system as a support to determine how to evolve the STS. We provide some examples from Table 4.

The objective *ATO* is hardly achieved in context *day-normal*, i.e., the degree of validity of the objective achievement assumption $\delta_O(ATO, \mathbf{dn})$ is below 0 (−0.273). This happens because of the higher number of vehicles driving during the day. The designer may therefore introduce different requirements to help achieve the objective, for instance, replacing the current centralized navigation service with a more intelligent one, or by changing the environment, e.g., by closing some roads to traffic.

Also, requirement *ANS* is harmful in context *night-normal*: the degree of validity of the requirement necessity assumption $\delta_N(ANS, \mathbf{O}, \mathbf{nn})$ is −0.7867, indicating that using *ANS* is detrimental to satisfying the objectives, which are quite positively satisfied when *ANS* is not employed. In the simulator, this happens for the adaptive navigation service uses some vehicles as "explorers" to find less congested roads (Schmid et al. 2017). This strategy appears to be harmful during the night since less vehicles drive in the city and roads are not congested. The designer may therefore disable the navigation service in such context.

The degrees of validity listed in the table can be also visualized directly on the original requirements model using a color overlay (see Dell'Anna et al. 2018a for an

Table 4 The degree of validity of the assumptions made in Fig. 3 in the four different operating contexts *day-normal weather* (**dn**), *day-extreme weather* (**de**), *night-normal weather* (**nn**), *night-extreme weather* (**ne**)

Assumption	**c** = **dn**	**c** = **de**	**c** = **nn**	**c** = **ne**
$\delta_S(NS, \mathbf{c})$	0.0580	0.1073	0.0473	0.0676
$\delta_S(NSD, \mathbf{c})$	0.1007	0.0983	0.0988	0.0946
$\delta_S(ANS, \mathbf{c})$	0	0	0	−0.0001
$\delta_S(SNS, \mathbf{c})$	0.1003	0.0980	0.0989	0.0940
$\delta_S(RS, \mathbf{c})$	0.0596	0.0598	0.0582	0.0581
$\delta_S(J, \mathbf{c})$	0	0	0	0
$\delta_S(SJ, \mathbf{c})$	0	0	0	0
$\delta_S(ATL, \mathbf{c})$	0	0	0	0
$\delta_S(STL, \mathbf{c})$	0	0	0	0
$\delta_S(P, \mathbf{c})$	0	0	0	0
$\delta_S(TR, \mathbf{c})$	0	0	0	0
$\delta_O(ATO, \mathbf{c})$	−0.2730	0.6591	0.9998	0.9999
$\delta_O(A, \mathbf{c})$	0.1119	−0.0379	0.8374	0.8181
$\delta_O(C, \mathbf{c})$	0.5079	0.5954	0.9605	0.9998
$\delta_O(D, \mathbf{c})$	0.9933	1	0.9998	0.9998
$\delta_O(T, \mathbf{c})$	0.3155	0.5604	0.7472	0.9737
$\delta_C(ATO, NS, \mathbf{c})$	−0.0744	−0.2470	0.0002	0
$\delta_C(ATO, J, \mathbf{c})$	0	0	0	0
$\delta_C(A, J, \mathbf{c})$	0	0	0	0
$\delta_C(C, NSD, \mathbf{c})$	−0.0137	0.3960	0.5105	0.4777
$\delta_C(D, NSD, \mathbf{c})$	0.4337	0.5454	0.5383	0.4998
$\delta_C(T, NSD, \mathbf{c})$	0.0704	0.3773	0.2412	0.5180
$\delta_{AND}(NS, \mathbf{c})$	0.0187	0.0226	0.0362	−0.0376
$\delta_{AND}(J, \mathbf{c})$	0	0	0	0
$\delta_{XOR}(NSD, \mathbf{c})$	0.1003	0.0980	0.0988	0.0940
$\delta_{XOR}(SJ, \mathbf{c})$	0	0	0	0
$\delta_{AD}(NS, NSD, \mathbf{c})$	−0.1008	−0.1033	−0.1008	−0.0019
$\delta_{AD}(NS, RS, \mathbf{c})$	0.0025	0.0043	0.0025	0.0025
$\delta_{AD}(NSD, ANS, \mathbf{c})$	0.3542	−0.1548	0.3542	−0.0253
$\delta_{AD}(NSD, SNS, \mathbf{c})$	1	1	1	0.6684
$\delta_{AD}(J, SJ, \mathbf{c})$	0	0	0	0
$\delta_{AD}(J, TR, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, ATL, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, STL, \mathbf{c})$	0	0	0	0
$\delta_{AD}(SJ, P, \mathbf{c})$	0	0	0	0
$\delta_N(NS, \mathbf{O}, \mathbf{c})$	−0.0068	−0.0029	−0.0022	0

Table 4 continued

Assumption	c = dn	c = de	c = nn	c = ne
$\delta_N(NSD, \mathbf{O}, \mathbf{c})$	−0.0040	−0.0056	0.0017	0.0029
$\delta_N(ANS, \mathbf{O}, \mathbf{c})$	−0.1003	−0.2498	−0.7867	−0.7336
$\delta_N(SNS, \mathbf{O}, \mathbf{c})$	−0.0033	−0.0036	−0.0005	0.0024
$\delta_N(RS, \mathbf{O}, \mathbf{c})$	−0.0018	0.0075	0.0018	−0.0015
$\delta_N(J, \mathbf{O}, \mathbf{c})$	0.1006	0.2502	0.7870	0.8967
$\delta_N(SJ, \mathbf{O}, \mathbf{c})$	−0.0005	−0.0008	0	0.0005
$\delta_N(ATL, \mathbf{O}, \mathbf{c})$	0.0093	−0.2501	−0.7865	−0.8968
$\delta_N(STL, \mathbf{O}, \mathbf{c})$	0.0004	0.0003	0	0.0003
$\delta_N(P, \mathbf{O}, \mathbf{c})$	0.0005	0.0013	−0.7864	0.0001
$\delta_N(TR, \mathbf{O}, \mathbf{c})$	0.1003	0.2501	0.7867	0.6191

example of such visualization). This may help the designer to quickly analyze the behavior of the system and to determine whether an intervention is required.

6 Automated requirements revision

In Sect. 5, we have described mechanisms for analysts to determine—assisted by an *RBN* that is populated with system execution logs—the validity of the assumptions that a requirements model implicitly contains. Such techniques help the analysts identify systems' behaviors that are not aligned with expectations, so that human evolution of the system requirements can be made.

Here, we present a control loop for the automated adaptation of an STS (Sect. 6.2), which leverages the information concerning assumptions validity learned at runtime, in order to revise the STS requirements aiming to maximize the system's objectives achievement. Prior to explaining the control loop, we define in Sect. 6.1 some key terms that concern our conceptual framework.

6.1 Requirement variant, system configuration, and requirement revision

The adaptation mechanisms presented in this section require the introduction of three basic notions: those of a *requirement variant* (Definition 5), *system configuration* (Definition 6) and *requirement revision* (Definition 7).

Definition 5 (*Requirement variant*) Consider a set $\mathcal{O}$ of stakeholders objectives in a requirements model $\mathcal{RM}$, and a set $\mathcal{C}$ of all possible contexts in which the system operates. We call *requirement variant* V a sub-graph of $\mathcal{RM}$ that is defined by pruning $\mathcal{RM}$ as follows:

1. for every XOR-refined requirement in $\mathcal{RM}$, V contains exactly one sub-requirement;

Table 5 The 12 requirement variants of the smart traffic scenario

Var.	Description	Requirements
$\mathcal{V}_1$	Static navigation system and static traffic lights	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *STL*, *TR*
$\mathcal{V}_2$	Adaptive navigation system and static traffic lights	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *STL*, *TR*
$\mathcal{V}_3$	Only static traffic lights	*J*, *SJ*, *STL*, *TR*
$\mathcal{V}_4$	Only priority lanes signs	*J*, *SJ*, *P*, *TR*
$\mathcal{V}_5$	All panels disabled	*J*, *TR*
$\mathcal{V}_6$	Only static navigation system	*NS*, *NSD*, *SNS*, *RS*, *J*, *TR*
$\mathcal{V}_7$	Only adaptive navigation system	*NS*, *NSD*, *ANS*, *RS*, *J*, *TR*
$\mathcal{V}_8$	Static navigation system and adaptive traffic lights	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_9$	Adaptive navigation system and adaptive traffic lights	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_{10}$	Only adaptive traffic lights	*J*, *SJ*, *ATL*, *TR*
$\mathcal{V}_{11}$	Static navigation system and priority lanes signs	*NS*, *NSD*, *SNS*, *RS*, *J*, *SJ*, *P*, *TR*
$\mathcal{V}_{12}$	Adaptive navigation system and priority lanes signs	*NS*, *NSD*, *ANS*, *RS*, *J*, *SJ*, *P*, *TR*

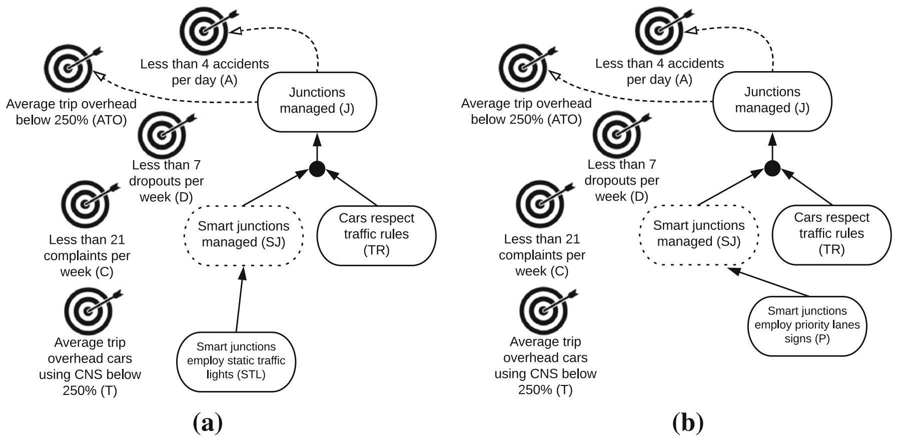

Less than 4 accidents per day (A)
Average trip overhead below 250% (ATO)
Junctions managed (J)
Less than 7 dropouts per week (D)
Less than 21 complaints per week (C)
Smart junctions managed (SJ)
Cars respect traffic rules (TR)
Average trip overhead cars using CNS below 250% (T)
Smart junctions employ static traffic lights (STL)
Smart junctions employ priority lanes signs (P)
(a)
(b)

Fig. 5 A graphical representation of the requirement variants $\mathcal{V}_3$ and $\mathcal{V}_4$

2. for every optional requirement in $\mathcal{RM}$, that requirement can either be included in or excluded from V.
3. if a requirement in $\mathcal{RM}$ is excluded from V through clauses 1. or 2., then all the descendants of that requirement are also pruned.

The requirements model of Fig. 3 results in a set $\mathcal{V}$ of twelve requirement variants (listed in Table 5) that satisfy the top-level requirement, computed by activating or disabling the optional requirements *NS* and *SJ*, and by making choices for the XOR-refined requirements *NSD* and *SJ*. Figure 5a and b report, as an example, a graphical representation of variants $\mathcal{V}_3$ and $\mathcal{V}_4$.

Definition 6 (System configuration) Given the set of requirement variants $\mathcal{V}$ and the set of operating contexts $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, a *system configuration* assigns a requirement variant to each operating context. Formally, a system configuration is a set of pairs $\{\langle \mathcal{C}_1, \mathcal{V}_i\rangle, \ldots, \langle \mathcal{C}_j, \mathcal{V}_k\rangle, \ldots, \langle \mathcal{C}_m, \mathcal{V}_p\rangle\}$ such that $\mathcal{V}_i, \mathcal{V}_k, \ldots, \mathcal{V}_p \in \mathcal{V}$.

Given the four possible contexts *day-normal*, *day-extreme*, *night-normal*, *night-extreme*, and given the set $\mathcal{V}$ of possible requirement variants, an example of system configuration is $\{\langle$*day-normal*, $\mathcal{V}_3\rangle$, $\langle$*day-extreme*, $\mathcal{V}_4\rangle$, $\langle$*night-normal*, $\mathcal{V}_5\rangle$, $\langle$*night-extreme*, $\mathcal{V}_{10}\rangle\}$.

A certain requirement R is said to be *active* in a context $\mathcal{C}_i$ if $\langle \mathcal{C}_i, \mathcal{V}_j\rangle$ is in the system configuration and $R \in \mathcal{V}_j$. Otherwise R is said *disabled*.

The concepts of requirement variant and system configuration are essential for us to define the notions of *requirement revision*, which is the basic action that the STS Supervisor performs when adapting the STS, on the basis of the learned runtime information concerning assumptions validity.

Definition 7 (*Requirement revision*) Given a requirements model $\mathcal{RM}$, and a requirement variant $\mathcal{V}_i$ of $\mathcal{RM}$, a revision of a requirement R with respect to $\mathcal{V}_i$ is an operation that returns a different variant $\mathcal{V}_j$ of $\mathcal{RM}$ with $i \neq j$. We distinguish the following types of revisions of a requirement R:

- *Disabling* $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ that does not contain R.[2]
- *Activating* $R \notin \mathcal{V}_i$ returns a $\mathcal{V}_j$ that contains R.
- *Relaxing* $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}' \subset \mathcal{D}$.
- *Strengthening* $R \in \mathcal{V}_i$ returns a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}' \supset \mathcal{D}$.
- *Altering* $R \in \mathcal{V}_i$ return a $\mathcal{V}_j$ such that, given the set $\mathcal{D}$ of descendants of R in $\mathcal{V}_i$, the set of descendants of R in $\mathcal{V}_j$ is $\mathcal{D}'$ such that $\mathcal{D}' \neq \mathcal{D}$ and $\mathcal{D}' \cap \mathcal{D} \neq \emptyset$.

Table 6 shows the revisions applied to each of the requirements (*NS*, *NSD*, etc.) in the requirements model of Fig. 3, in order to obtain the twelve possible requirement variants starting from $\mathcal{V}_1$. For example, the requirement variant $\mathcal{V}_3$ (see Table 5) is obtained from $\mathcal{V}_1$ by *disabling* the requirement *NS* and by consequence also all of its descendants. On the other hand, the requirement variant $\mathcal{V}_2$ is obtained from $\mathcal{V}_1$ by *altering NS* by replacing the descendant requirement *SNS* with *ANS*. In requirement variant $\mathcal{V}_5$, the requirement J is *relaxed* w.r.t. variant $\mathcal{V}_1$, for its descendants *SJ* and *STL* are in $\mathcal{V}_1$ but not in $\mathcal{V}_5$. Notice that if we had started from $\mathcal{V}_5$ instead (not shown in the table), the same requirement J would have been *strengthened* in variant $\mathcal{V}_1$. Finally, requirement P is *activated* in $\mathcal{V}_4$, for it was not present in $\mathcal{V}_1$.

Definition 8 lifts the notion of revision from an individual requirement (Definition 7) to an entire requirement variant.

Definition 8 (*Requirement variant revision*) Given a requirements model $\mathcal{RM}$, and given two requirement variants $\mathcal{V}_i, \mathcal{V}_j$ of $\mathcal{RM}$, $\mathcal{V}_j$ is a revision of $\mathcal{V}_i$ if and only if $\mathcal{V}_j \neq \mathcal{V}_i$. A requirement variant revision can be of three types:

- *Relaxation*: for each requirement R in $\mathcal{RM}$, either R is not revised between $\mathcal{V}_i$ and $\mathcal{V}_j$, or it is relaxed or disabled;

[2] Clause 3 of Definition 5 ensures that all the descendants of R in $\mathcal{RM}$ are also not in $\mathcal{V}_j$.

Table 6 Revisions of the requirements in Fig. 3 that are performed from requirement variant $\mathcal{V}_1$ to the other eleven requirement variants

	$\mathcal{V}_1$	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	$\mathcal{V}_6$	$\mathcal{V}_7$	$\mathcal{V}_8$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$
$\underline{NS}$	–	alt	dis	dis	dis	–	alt	–	alt	dis	–	alt
$\underline{NSD}$	–	alt	dis	dis	dis	–	alt	–	alt	dis	–	alt
$\underline{SNS}$	–	dis	dis	dis	dis	–	dis	–	dis	dis	–	dis
ANS	–	act	–	–	–	–	act	–	act	–	–	act
$\underline{RS}$	–	–	dis	dis	dis	–	–	–	–	dis	–	–
$\underline{J}$	–	–	–	alt	rel	rel	rel	alt	alt	alt	alt	alt
$\underline{SJ}$	–	–	–	alt	dis	dis	dis	alt	alt	alt	alt	alt
$\underline{STL}$	–	–	–	dis	dis	dis	dis	dis	dis	dis	dis	dis
ATL	–	–	–	–	–	–	–	act	act	act	–	–
P	–	–	–	act	–	–	–	–	–	–	act	act
$\underline{TR}$	–	–	–	–	–	–	–	–	–	–	–	–
$\mathcal{V}_1$	–	alt	rel	alt	rel	rel	alt	alt	alt	alt	alt	alt

The requirements whose nodes are in $\mathcal{V}_1$ are underlined. The last row describes the variant revision type from $\mathcal{V}_1$ to the other requirement variants. Value "–" indicates that no revision is applied

- *Strengthening*: for each requirement R in $\mathcal{RM}$, either R is not revised between $\mathcal{V}_i$ and $\mathcal{V}_j$, or it is strengthened or activated;
- *Alteration*: when $\mathcal{V}_j$ is neither a relaxation or a strengthening of $\mathcal{V}_i$.

The last line of Table 6 determines the type of variant revision based on the individual requirements revisions. For example, $\mathcal{V}_3$ is a relaxation of $\mathcal{V}_1$, for the only requirement revision type that is applied is disabling (*NS*, *NSD*, *SNS*, and *RS*). $\mathcal{V}_2$ is instead an alteration of $\mathcal{V}_1$, for the applied requirement revisions do not define neither a relaxation nor a strengthening.

6.2 The STS supervisor control loop

The notions introduced in Definitions 5–8 are used to explain the *STS Supervisor* (first mentioned in Fig. 1) that guides the adaptation of an STS. The control loop of the STS Supervisor is shown in Fig. 6 and described in the following.

At design-time, an initial system configuration (as per Definition 6) is selected by the analyst according to the available domain knowledge, and it is stored in the *System Configuration* component.

At runtime, the *Monitoring* component collects information about the satisfaction or violation of the requirements and about the operating contexts in which they are evaluated. The overall objectives are also evaluated, typically with lower frequency and relying on aggregate information. This knowledge (the *STS log*) is used to learn, by means of a *Requirement Bayesian Network* (described in Sect. 2.2), correlations between the satisfaction of the requirements and the achievement of the objectives in the different contexts.

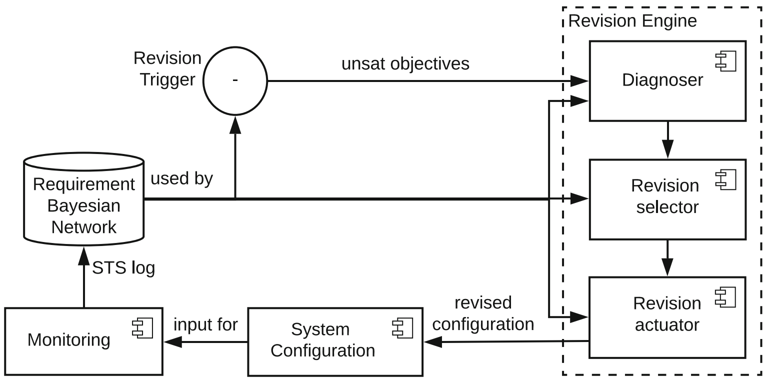


Fig. 6 The main components of the STS Supervisor

A *Revision Trigger* component (Sect. 6.2.1) uses the learned knowledge to determine whether some requirements should be revised. The requirements revision process is executed by the *Revision Engine* component that generates as output a (possibly) new system configuration, replacing the current one in the *System Configuration* component. The sub-components of the *Revision Engine* are detailed in Sects. 6.2.2–6.2.4.

The STS Supervisor control loop implements a variant of the hill climbing optimization algorithm. A system configuration is treated as a solution in the space of all possible solutions. We say that the hill climbing optimization process (Supervisor's control loop) performs a *step* every time a requirement revision process is triggered by the *Revision Trigger*. A new solution (system configuration) is then selected among the solutions in the neighborhood of the current system configuration. The neighborhood of a system configuration is defined by our *Revision Engine* component by making use of the requirements model's structure and of the *RBN*. In particular, in Sect. 6.2.3 we describe two different algorithms for the selection of a requirements' revision that can be used as informed heuristics for the definition of a neighborhood of a system configuration. In Sect. 7 we will then evaluate such heuristics by comparing them with uninformed ones that do not leverage runtime execution data about the validity of the assumptions underlying a requirements model.

6.2.1 Revision trigger

The Revision Trigger determines if a requirements revision is necessary. If so, the *Diagnoser* (Sect. 6.2.2) is invoked; otherwise, no revision of requirements is triggered.

Let e be an event representing network stability: changes in the probability distributions in the *Requirement Bayesian Network* are not significant anymore (i.e., the variations in the distribution when a new sample is given are below a specified threshold). Assuming a consistent behavior of the system, such event will occur after some time.

Let t_{oa} be a threshold defining the minimum objectives achievement joint probability (i.e., the probability oa that all the objectives are achieved together) desired by the system designer. A revision (i.e., a new step of the hill climbing procedure) is triggered every time e occurs and t_{oa} is not met with the current system configuration (i.e., $oa < t_{oa}$ with the current system configuration). For example, $oa \geq 0.95$ indicates a threshold t_{oa} of 95% for the objectives achievement joint probability.

Revisions are triggered based on an analysis of the *objective achievement assumptions*. The revision trigger calculates the joint degree of validity of the objective achievement assumption for all the objectives. A revision is triggered when such value is below $t_{oa} - (1 - t_{oa})$.

Please note that, as above described, at any time instant a certain system configuration $\mathcal{C}_i$ is chosen. The objectives achievement joint probability oa, therefore, depends on the chosen configuration. For instance, for the running example, if $\mathcal{C}_i = \{\langle \textit{day-normal}, \mathcal{V}_3\rangle, \langle \textit{day-extreme}, \mathcal{V}_4\rangle, \langle \textit{night-normal}, \mathcal{V}_5\rangle, \langle \textit{night-extreme}, \mathcal{V}_{10}\rangle\}$, then oa needs to be calculated as follows:

$$\begin{aligned} oa = {} & P(\mathbf{O}_{true} |\, \mathbf{dn} \ \wedge \ \mathbf{v3}) P(\mathbf{dn}) + P(\mathbf{O}_{true} |\, \mathbf{de} \ \wedge \ \mathbf{v4}) P(\mathbf{de}) \\ & + P(\mathbf{O}_{true} |\, \mathbf{nn} \ \wedge \ \mathbf{v5}) P(\mathbf{nn}) + P(\mathbf{O}_{true} |\, \mathbf{ne} \ \wedge \ \mathbf{v10}) P(\mathbf{ne}) \end{aligned} \tag{10}$$

with **dn**, **de**, **nn**, **ne** evidences for the contexts *day-normal*, *day-extreme*, *night-normal*, *night-extreme*, respectively (e.g., $\mathbf{dn} = (Time_{day} \wedge Weather_{normal})$), and **v3**, **v4**, **v5**, **v10** evidences for the values of the requirements in the requirement variants $\mathcal{V}_3, \mathcal{V}_4, \mathcal{V}_5, \mathcal{V}_{10}$, respectively (e.g., $\mathbf{v3} = (J_{act} \wedge SJ_{act} \wedge STL_{act} \wedge TR_{act} \wedge \mathbf{V3D}_{dis})$, where **V3D** is the set of remaining requirement nodes disabled in $\mathcal{V}_3$).

Analogously, any probability that needs to be calculated on the *Requirement Bayesian Network* w.r.t. a certain context should take into account the currently chosen system configuration. In order to ease the reading, however, in the rest of the paper we do not explicitly represent (unless differently specified) the evidence for the requirement nodes. We implicitly assume, instead, that in a certain context **c** the given evidence informs also about the active/disabled requirements in the requirement variant currently chosen for context **c**. For instance, if $\mathcal{V}_3$ is currently chosen for context **c**, and we want to calculate the objectives achievement joint probability in such context **c**, instead of writing $P(\mathbf{O}_{true} |\, \mathbf{c} \wedge \mathbf{v3})$ (with **v3** defined as above) we simply write $P(\mathbf{O}_{true} |\, \mathbf{c})$.

6.2.2 Diagnoser

When a revision is triggered, the *Diagnoser* component is invoked to determine the reasons why the objectives are not achieved. To do so, it uses the *Requirement Bayesian Network* to determine the most problematic operating context in which the objectives are not achieved.

Let **all** be the set of all possible assignments of a value to each of the context variables in the Bayesian Network (e.g., for the Bayesian Network reported in Fig. 4, $\mathbf{all} = \{\{Time_{day}, Weather_{normal}\}, \ldots, \{Time_{night}, Weather_{extreme}\}\}$). The most problematic context (denoted with **mpc**) is the assignment resulting from Eq. 11.

$$\mathbf{mpc} = argmax_{\mathbf{c} \in \mathbf{all}} P(\mathbf{O}_{false} \mid \mathbf{c}) \tag{11}$$

6.2.3 Revision selector

Let $\mathcal{V}_{\mathbf{mpc}}$ be the requirement variant assigned to the most problematic context **mpc** in the current system configuration. The *Revision Selector* determines the most adequate requirements revisions to perform to requirements in $\mathcal{V}_{\mathbf{mpc}}$ so to increase $P(\mathbf{O}_{true} \mid \mathbf{mpc})$. Our framework includes two heuristic algorithms: PUREBN (PB) and STATEBASED (SB). We report here their working principles, and refer the reader to Dell'Anna et al. (2018b) for more details.

PB and SB first identify the relationship between the requirement and the objective nodes in the Bayesian Network by performing an analysis of some of the design-time assumptions described in Sect. 5.

Requirements can be either *useful* for the achievement of the overall objectives or *harmful*. Useful requirements can be further divided into requirements that are *more useful when obeyed* and requirements that are *more useful when violated*. Useful requirements can also be either *often obeyed* when the objectives are not achieved or *often violated*.

Let us formalize this classification in terms of probability theory. Let $\mathbf{R}$ be the set of all requirement nodes in a *Requirement Bayesian Network*. In the rest of this section, for simplicity, let $\mathbf{R} = \{X, Y, Z\}$.

Harmful requirements The set of requirements such that, when all disabled, guarantee a better objectives achievement joint probability than when at least one of them is activated. Let **da** be the set of all possible assignments of values in the set $\{dis, act\}$ to all nodes $\mathbf{R}$ (e.g., given $\mathbf{R} = \{X, Y, Z\}$, then $\mathbf{da} = \{\{X_{dis}, Y_{dis}, Z_{dis}\}, \ldots, \{X_{act}, Y_{act}, Y_{act}\}\}$).

Let **h** be the assignment of Eq. 12 (e.g., $\mathbf{h} = \{X_{dis}, Y_{act}, Z_{act}\}$).

$$\mathbf{h} = argmax_{\mathbf{r} \in \mathbf{da}}\ P(\mathbf{O}_{true} | \mathbf{r}\ \wedge \mathbf{mpc}) \tag{12}$$

Let $\mathbf{D} \subseteq \mathbf{R}$ be the set of nodes that have value *dis* in **h**, and **A** be the set $\mathbf{R} \setminus \mathbf{D}$ (e.g., $\mathbf{D} = \{X\}$ and $\mathbf{A} = \{Y, Z\}$). Harmful requirements are all the requirements such that the corresponding nodes in the Bayesian Network are in **D**. Useful requirements are instead all the requirements such that the corresponding nodes in the Bayesian Network are in **A**.

Note that an harmful (useful) requirement is one whose associated *requirement necessity assumption* is negative (positive).

Requirements that are more useful when obeyed (violated) The set of requirements that are most useful for the objectives achievement joint probability when active, either when obeyed or violated.

Let **ov** be the set of all possible assignments of values in the set $\{ob, viol\}$ to all nodes in the set of useful requirements **A** (e.g., given $\mathbf{A} = \{Y, Z\}$, then $\mathbf{ov} = \{\{Y_{ob}, Z_{ob}\}, \{Y_{ob}, Z_{viol}\}, \{Y_{viol}, Z_{ob}\}, \{Y_{viol}, Z_{viol}\}\}$). Let **u** be the assignment resulting from Eq. 13 (e.g., $\mathbf{u} = \{Y_{ob}, Z_{viol}\}$).

$$\mathbf{u} = argmax_{\mathbf{r} \in \mathbf{ov}}\ P(\mathbf{O}_{true} | \mathbf{r}\ \wedge \mathbf{mpc} \wedge \mathbf{D}_{dis}) \tag{13}$$

Requirements that are more useful when obeyed (violated) are all the requirements whose nodes in the Bayesian Network have value *ob* (*viol*) in **u** (e.g., Y is more useful when obeyed, while Z is more useful when violated).

Determining whether a requirement is useful when obeyed or when violated corresponds to evaluate if the associated *contribution assumption* is positive or negative. When a requirement has no *aims at* link to an objective in $\mathcal{RM}$, we are evaluating a hypothetical link between the two elements.

Useful requirements often obeyed (violated) when $\mathbf{O}_{false}$ The set of useful requirements that are most likely to be obeyed (violated) when the objectives are not achieved. Let **mle** be the assignment of Eq. 14 (e.g., $\mathbf{mle} = \{Y_{ob}, Z_{ob}\}$). We call such assignment *most likely explanation* for $\mathbf{O}_{false}$ in **mpc**.

$$\mathbf{mle} = argmax_{\mathbf{r}\in\mathbf{ov}}\ P(\mathbf{r}\ |\mathbf{O}_{false} \wedge \mathbf{mpc} \wedge \mathbf{D}_{dis}) \tag{14}$$

Useful requirements that are often obeyed (violated) when $\mathbf{O}_{false}$ are those whose corresponding nodes in the *Requirement Bayesian Network* have value *ob* (*viol*) in **mle** (e.g., both nodes Y and Z are often obeyed when $\mathbf{O}_{false}$).

The most likely explanation **mle** for $\mathbf{O}_{false}$ in **mpc** is determined by computing the most likely degree of validity of the *requirement satisfiability assumptions* of the requirements that are not harmful. The most likely value of a requirement R is *obeyed* if the most likely degree of validity $\delta_S(R, (\mathbf{mpc}, \mathbf{O}_{false}))$ is positive, otherwise the most likely value is *violated*.

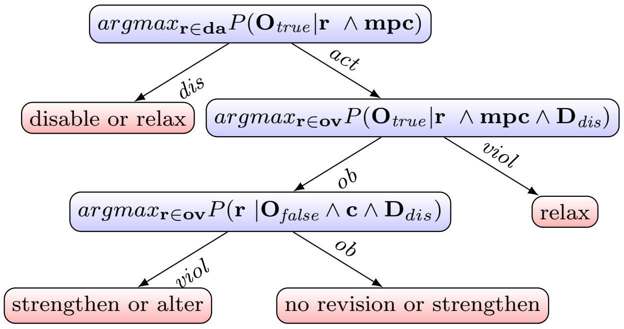
Algorithm PB After identifying the relationship between requirements and overall objectives, as just described, this algorithm applies the following procedure, also illustrated by the decision tree of Fig. 7:

1. *Disable*/*Relax* harmful requirements.
2. *Relax* useful requirements that are more useful when violated.
3. *Strengthen*/*Alter* useful requirements that are more useful when obeyed but they are often violated when $\mathbf{O}_{false}$.
4. Keep all other requirements unrevised, or *strengthen* them.

For example, given **R** as above described, PB suggests to disable/relax X, to relax Z and to either leave unaltered Y or to strengthen it.

Algorithm SB This algorithm implements a different strategy for the revision selection, for it analyzes the relationship between the *average* requirements satisfaction (calculated as the mean) and the objectives achievement joint probability of the current system configuration. Figure 8 plots five examples of system configurations in four states with respect to the average requirements satisfaction and the objectives achievement joint probability.

Configurations in state A sufficiently satisfy the requirements, but this does not lead to sufficient objectives achievement. State B has insufficient requirements satisfaction and objectives achievement. State C indicates that the objectives are achieved even though the requirements are not satisfied. State D is the ideal area: the requirements



$argmax_{\mathbf{r}\in\mathbf{da}} P(\mathbf{O}_{true}|\mathbf{r} \wedge \mathbf{mpc})$
dis
act
disable or relax
$argmax_{\mathbf{r}\in\mathbf{ov}} P(\mathbf{O}_{true}|\mathbf{r} \wedge \mathbf{mpc} \wedge \mathbf{D}_{dis})$
ob
$viol$
$argmax_{\mathbf{r}\in\mathbf{ov}} P(\mathbf{r}\ |\mathbf{O}_{false} \wedge \mathbf{c} \wedge \mathbf{D}_{dis})$
relax
$viol$
ob
strengthen or alter
no revision or strengthen

Fig. 7 Decision tree used by algorithm PB for determining a suitable type of revision

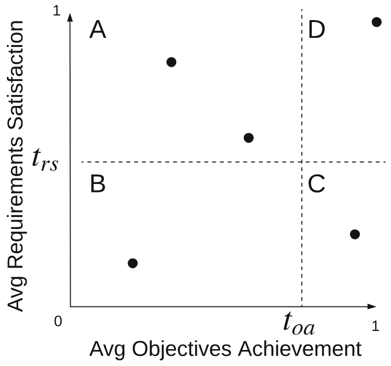


Fig. 8 Plotting system configurations (points) in four states (A–D) according to the average requirements satisfaction and objectives achievement joint probability. t_{rs} and t_{oa} denote the desired average requirements satisfaction and objectives achievement joint probabilities, respectively

are satisfied and the objectives are achieved. Algorithm SB aims to revise the system configuration and move the system into state D by applying the following procedure:

1. Calculate average requirements satisfaction probability.
2. Calculate objectives achievement joint probability.
3. *Disable* harmful requirements, if any. Else, go to point 4.
4. If the system configuration is in state A: *Relax* useful requirements that are more useful when violated but often obeyed when $\mathbf{O}_{false}$, if any. Otherwise, *Strengthen*/*Alter* all useful requirements.
5. If the system configuration is in state B: *Strengthen*/*Alter* useful requirements that are more useful when obeyed but often violated when $\mathbf{O}_{false}$ and *Relax* useful requirements that are more useful when violated.
6. If the system configuration is in state C: *Relax* useful requirements that are more useful when violated and often violated when $\mathbf{O}_{false}$, if any. Otherwise, *Strengthen*/*Alter* useful requirements that are more useful when obeyed but often violated when $\mathbf{O}_{false}$.

For example, given $\mathbf{R}$ as described above, SB only suggests to disable X, for such requirement is *harmful*.

PB and SB adopt different strategies for the revision of requirements. While PB determines for all the requirements the most opportune revision to perform (if any), SB considers the global state of the system and suggests to revise only a certain type of requirements at every iteration. This difference leads to a different definition of the neighborhoods of the configurations during the hill climbing process (see Sect. 6.2.4), and this leads to different results (as will be visible in Sect. 7). We refer the reader to our previous work (Dell'Anna et al. 2018b) for more details about the implementation of the algorithms.

6.2.4 Revision actuator

This component adopts a new requirement variant in the **mpc**. Given a list of suggested revisions for requirements in the requirement variant $\mathcal{V}_{\mathbf{mpc}}$ currently assigned to context **mpc** in the system configuration, the Revision Actuator selects a requirement variant $\mathcal{V}_j$ that is as much aligned as possible with the direction provided by the suggestion.

For example, consider $\mathcal{V}_1$, and assume the Revision Selector suggests to alter the requirement *NS*. Then, the Revision Actuator has to find other requirement variants where *NS* is altered from $\mathcal{V}_1$ (e.g., $\mathcal{V}_2$, $\mathcal{V}_7$, $\mathcal{V}_9$ or $\mathcal{V}_{12}$, see Table 6). The obtained set of variants defines the neighborhood of the current system configuration in the hill climbing optimization process.

If the neighborhood contains multiple variants, different distance metrics can be defined, e.g., the similarity with the current variant, or the sensitivity of the objectives to the change of the selected requirements. Here, we adopt the number of revisions of requirements needed to obtain $\mathcal{V}_j$ from $\mathcal{V}_{\mathbf{mpc}}$. For instance, four revisions are necessary to obtain $\mathcal{V}_2$ from $\mathcal{V}_1$, seven revisions are necessary to obtain $\mathcal{V}_7$ from $\mathcal{V}_1$, etc.

After selecting the new requirement variant $\mathcal{V}_j$, the current system configuration is updated to map the context **mpc** to the new $\mathcal{V}_j$ instead of $\mathcal{V}_{\mathbf{mpc}}$.

If there is no new variant that is aligned with the provided suggestion (i.e., the neighborhood is empty or it contains only already-attempted variants), the Revision Actuator randomly selects a system configuration never tried before, if any. This makes our implementation of hill climbing different from traditional ones, and guarantees convergence to an optimal solution.

7 Evaluation

We report on an evaluation of the proposed supervision mechanism; in particular, we conduct an experiment that investigates the process through which the Supervisor's control loop identifies an optimal system configuration.

7.1 Scope, context, and hypotheses

The object of our study consists of the requirement revision heuristics. We compare two sets of dependent variables:

i. *Informed heuristics*: the two algorithms PB and SB described in Sect. 6.2.3 implemented in our *Revision Engine*; and

ii. *Uninformed heuristics*: three baseline algorithms that do not leverage knowledge about the validity of the design-time assumptions:

   1. Maximum distance 8 (D8) defines a neighbourhood composed of all the system configurations that are obtained by revising at most 8 requirements;[3]
   2. Maximum size 10 (S10) defines a neighborhood composed of the 10 closest system configurations to the current one;
   3. Maximum size 20 (S20) defines a neighborhood composed of the 20 closest system configurations to the current one.

We identify four independent variables for studying the process through which the Supervisor's control loop identifies an optimal system configuration:

1. *Convergence speed*: the number of steps (i.e., revisions triggered by the *Revision Trigger*, as described in Sect. 6.2) and the number of explored system configurations that the Supervisor's control loop requires before it identifies an optimal system configuration;
2. *Quality*: the probability that the system configurations explored satisfy the system objectives;
3. *Stability*: the number of requirements revisions that are performed while identifying an optimal system configuration.

Furthermore, for the informed algorithms alone, we evaluate *4. Noise tolerance*: the degree to which the amount of noisy input data (imperfect monitors) affects convergence speed, quality, and stability. Noise tolerance does not affect uninformed algorithms, for they do not take into account any information about requirements satisfaction.

Our experiment is run through CrowdNavExt, the simulation environment that instantiates the smart traffic example presented in Sect. 3. Within the context of such simulation environment, we formulate the following hypotheses:

- H_1: our informed heuristics provide a higher convergence speed than the uninformed heuristics;
- H_2: our informed heuristics allow to higher-quality system configurations than the uninformed heuristics;
- H_3: our informed heuristics allow to perform less revisions than the uninformed heuristics while finding an optimal system configuration;
- H_4: noisy input data has a marginal effect on convergence speed, quality, and stability of the Supervisor's control loop when using our informed heuristics.

7.2 Design and instrumentation

SASS (Supervisor of Autonomous Software Systems)[4] is our implementation of the Supervisor's control loop described in Sect. 6 as a modified version of hill climbing.

[3] The value of 8 was chosen via experimentation with CrowdNav. Revising one requirement leads to a distance of 4–5 from the original system configuration, and each system configuration has 10–20% of all system configurations in its neighborhood.

[4] SASS' code repository: https://bitbucket.org/dellannadavide/sass.

The supervisor performs a local search and stops when either (i) all the system configurations have been tried; or (ii) a local optimum (system configuration) is found that has objectives achievement joint probability *oa* above the desired threshold t_{oa}. This probability is determined from simulation data (see Sect. 6.2.1) as the joint probability of achievement of all the objectives, given the chosen system configuration. We call optimal the last system configuration chosen, since either it is above the desired threshold or there is no other better configuration.

CrowdNavExt has $12^4 = 20,736$ possible system configurations, i.e., assignments of one of the twelve variants to each of the four contexts (see Definition 6). To keep our simulation time manageable, we chose 81 system configurations via test case generation techniques. We first applied *pairwise testing*: for each pair of variables, we obtained all their possible discrete combinations. Our variables are: time of the day (day, night), weather (normal, extreme), the alternative requirements for the navigation service (none, adaptive, static), and the alternatives for managing smart junctions (none, adaptive lights, static lights, priority lanes). This led to 3 different variants for each of the 4 operating contexts, using pairwise testing. We generated all combinations of the four groups of variants (each system configuration includes four variants, one per each operating context). Finally, we introduced three additional system configurations more distant from the others (in terms of number of required revisions, as described in Sect. 6.2.4). Two of them are the best-scoring system configurations. Therefore, in our experiments, we study 84 system configurations (reported in Table 7).

Table 8 describes all the simulation parameters of our experiments. We run simulations with three possible values of t_{oa}: 0.35, 0.3, and 0.25. These values have been determined manually, based on the objectives achievement joint probability *oa* of the 84 system configurations (shown in Fig. 9), so that the three different values determine, as also reported in Table 8, three levels of difficulty for the search of an optimal configuration in terms of percentage of system configurations above the threshold.

We test SASS with the two informed algorithms described in Sect. 6 (PB and SB) as heuristics for defining the neighborhood of a system configuration, i.e., the set of all the other system configurations that satisfy the suggestions provided by the suggestion selector with the help of the trained Bayesian Network. Moreover, we test the three additional uninformed heuristics D8, S20, S20 above described for use as baseline. Note that, in terms of Supervisor's control loop, the uninformed heuristics differ from the informed ones in that they do not use the *Revision Engine* to select a new system configuration. Since the rest of the control loop is common, for the sake of readability, we mention a certain heuristic algorithm (e.g., SB) to refer to the version of the Supervisor's control loop that uses such algorithm (e.g., the Supervisor's control loop with the SB heuristic).

In order to obtain significant data, due to the stochastic nature of the simulation data, SASS has been executed starting from all of the 84 possible system configurations with all the five tested heuristics.

We use the following metrics to determine whether our hypotheses hold:

- *Convergence speed* (H_1, detailed in Sect. 7.3.1)
    1. *Number of steps*: the average number of steps that an algorithm attempts before stopping (i.e., before finding a configuration above the desired threshold t_{oa}).

Table 7 The 84 system configurations employed for the experiments

Conf	**nn**	**dn**	**ne**	**de**	conf	**nn**	**dn**	**ne**	**de**	conf	**nn**	**dn**	**ne**	**de**
1	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	29	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	57	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
2	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	30	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	58	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
3	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	31	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	59	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
4	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	32	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	60	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
5	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	33	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	61	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$
6	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	34	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$	62	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$
7	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_8$	35	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$	63	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$
8	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_1$	36	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$	64	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$
9	$\mathcal{V}_7$	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_5$	37	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	65	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$
10	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	38	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	66	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
11	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	39	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	67	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
12	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	40	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	68	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
13	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	41	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	69	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
14	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	42	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	70	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$
15	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	43	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$	71	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$
16	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_8$	44	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$	72	$\mathcal{V}_6$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$
17	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_1$	45	$\mathcal{V}_2$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$	73	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$
18	$\mathcal{V}_7$	$\mathcal{V}_9$	$\mathcal{V}_4$	$\mathcal{V}_5$	46	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	74	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$
19	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	47	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	75	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$
20	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	48	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	76	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$
21	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{12}$	$\mathcal{V}_5$	49	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	77	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$
22	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_8$	50	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	78	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$
23	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_1$	51	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	79	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$
24	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_{10}$	$\mathcal{V}_5$	52	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$	80	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$
25	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_8$	53	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$	81	$\mathcal{V}_6$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$
26	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_1$	54	$\mathcal{V}_2$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	82	$\mathcal{V}_1$	$\mathcal{V}_3$	$\mathcal{V}_5$	$\mathcal{V}_8$
27	$\mathcal{V}_7$	$\mathcal{V}_3$	$\mathcal{V}_4$	$\mathcal{V}_5$	55	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	83	$\mathcal{V}_3$	$\mathcal{V}_2$	$\mathcal{V}_5$	$\mathcal{V}_8$
28	$\mathcal{V}_2$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_8$	56	$\mathcal{V}_6$	$\mathcal{V}_{11}$	$\mathcal{V}_{12}$	$\mathcal{V}_1$	84	$\mathcal{V}_{11}$	$\mathcal{V}_4$	$\mathcal{V}_6$	$\mathcal{V}_3$

nn, **dn**, **ne**, **de** respectively represent the contexts *night-normal*, *day-normal*, *night-extreme* and *day-extreme*

2. *Number of explored configurations*: the average percentage of system configurations that an algorithm attempts before stopping.

- *Quality* (H_2, detailed in Sect. 7.3.2)

1. *Final conf*: the average objectives achievement joint probability (i.e., the average *oa*) of the final solutions determined by an algorithm.
2. $\overline{oa}_A$: the average *oa* of all the configurations tried by an algorithm *A* before stopping.
3. $\overline{oa}_{A,last}$: the average *oa* of all the configurations tried by an algorithm *A* until all algorithms terminate all the 84 executions. Note that, if *A* terminates before

Table 8 Simulation parameters for our experiment with CrowdNavExt

Parameter	Value	Description
Time	day	600 vehicles in the city
	night	300 vehicles in the city
Weather	normal	Speed limits as per CrowdNav
	extreme	Speed limits reduced by 25%
Objective achievement threshold	0.35	3.5% system configurations above the threshold
	0.3	7% system configurations above the threshold
	0.25	17.8% system configurations above the threshold
Hill climbing heuristic	D8	Uninformed, neighbors max 8 revised revisions
	S10	Uninformed, up to 10 neighbors
	S20	Uninformed, up to 20 neighbors
	PB	Informed, PUREBN
	SB	Informed, STATEBASED

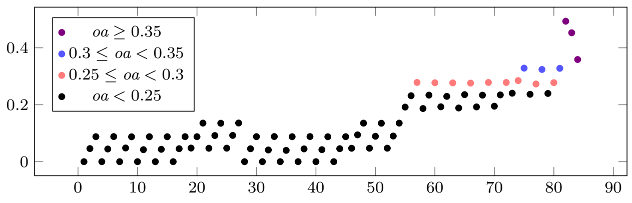


Fig. 9 Objectives achievement joint probability (y-axis) for all the 84 system configurations (x-axis)

other algorithms, the average will count, for the remaining steps, the *oa* of the last (optimal) configuration found.

4. $\overline{oa}_{A,first}$: the average *oa* of all the configurations tried by an algorithm *A* until the fastest algorithm terminates all the 84 executions (note that *A* is not necessary the fastest algorithm).
5. $\overline{oa}_{A,thres}$: the average *oa* of all the configurations tried by an algorithm *A* until the fastest algorithm reaches the threshold t_{oa} (note that this does not necessary mean that the fastest algorithm has terminated all its executions, nor that algorithm *A* has reached the threshold).

– *Stability* (H_3, detailed in Sect. 7.3.3)

1. *Revisions per step*: the average number of requirements revisions performed by an algorithm at each step before to reach the final solution.
2. *Total revs*: the average total number of revisions performed for a given algorithm to reach the final solution.

– *Noise tolerance* (H_4, only for PB and SB and detailed in Sect. 7.4): the variation in performance (speed, quality, stability) when noisy data concerning requirements satisfaction are used to train the *Requirement Bayesian Network*.

In the rest of the section, we present and discuss the results obtained in our experimentation w.r.t. the metrics and hypotheses described above.

7.3 Informed versus uninformed heuristics: speed, quality, and stability

Table 9 summarizes the results concerning H_1–H_3 obtained with the five tested heuristics. The table presents the results for the three tested thresholds of t_{oa} and reports the values (average and standard deviation) obtained from the 84 simulation runs for each of the metrics described in the previous section. As stated earlier, the baseline uninformed heuristics are denoted as D8, S10 and S20, while our informed heuristics are denoted as PB and SB.

7.3.1 Convergence speed (H_1)

Number of steps With all the thresholds, our informed heuristics consistently outperform the uninformed algorithms in terms of number of steps: see the *# steps* column of Table 9 and the bar chart of Fig. 10, both reporting the average number of steps that each algorithm attempted before stopping.

With $t_{oa} = 0.35$, the three uninformed heuristics D8, S10, S20 take on average 67.8 steps. Our PB and SB heuristics, instead, explore on average only 43.05 system configurations. In this scenario, the few (3 out of 84) optimal system configurations are slightly more distant (in terms of number of necessary revisions) from the non-optimal ones: while the average distance between the 81 system configurations is 16, that with the remaining 3 is 20. This affects the number of steps required to find one of them, for all the algorithms give priority to the closest system configurations in the neighborhood. Despite this difficulty, however, PB and SB deliver an improvements of $36.5\% = 1 - (43.05/67.8)$ over the uninformed heuristics in terms of required steps.

With $t_{oa} = 0.3$, the improvement over uninformed heuristics is even higher: $55.7\% = 1 - (13.4/30.27)$, and the efficiency gain increases further with $t_{oa} = 0.25$: $75.5\% = 1 - (3.69/15.09)$. Notably, with this last threshold, SB requires on average only 1.77 steps, i.e., it finds an optimal system configuration after only one or two revisions.

In Fig. 11, we show the percentage of steps required by the algorithms in order to terminate the first, second and third quartiles (respectively 25, 50 and 75%) of the 84 execution and the first 95% of them. It is worth noting that, in the case of $t_{oa} = 0.35$ (and similarly for the other thresholds), SB terminates the 95% of all the executions before any other uninformed heuristic terminates the first quartile.

When we compare our two informed heuristics, SB outperforms PB. PB suggests different revision types for different requirements. The selection of a requirement variant that satisfies the given suggestions, however, depends on the number of available variants (only 12 in our working example). The suggestions of SB affect, instead, requirements in the same quadrant of Fig. 8, thereby moving the current system configuration step-by-step toward the high requirements satisfaction and high objective achievement area. This strategy, which in almost all cases proved to be very efficient,

Table 9 Comparison of the algorithms with the three objective achievement thresholds

A	Speed				Quality										Stability			
	# steps		% explored configs		Final conf		$\overline{oa}_A$		$\overline{oa}_{A,last}$		$\overline{oa}_{A,first}$		$\overline{oa}_{A,thres}$		Revisions per step		Total revs	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
$t_{oa} = 0.35$																		
D8	75.67	15.79	0.76	0.15	**0.49**	0.02	0.17	0.11	0.22	0.10	0.18	0.03	0.18	0.04	3.84	0.76	301.29	63.95
S10	63.82	15.09	0.68	0.15	**0.49**	0.02	0.16	0.11	0.26	0.13	0.21	0.08	0.17	0.04	3.96	0.79	262.02	64.62
S20	64.05	14.10	0.72	0.16	**0.49**	0.02	0.17	0.11	0.26	0.14	0.21	0.09	0.16	0.05	4.36	0.88	289.24	63.30
PB	48.56	15.74	0.49	0.16	0.49	0.03	**0.18**	0.11	0.32	0.12	0.28	0.10	**0.20**	0.02	3.28	0.74	168.90	66.66
SB	**37.63**	17.19	**0.42**	0.18	**0.49**	0.02	0.17	0.11	**0.36**	0.13	**0.33**	0.12	0.20	0.06	**2.41**	0.56	**97.68**	53.47
$t_{oa} = 0.3$																		
D8	23.77	10.91	0.27	0.12	0.33	0.02	0.16	0.10	0.26	0.08	0.18	0.06	0.17	0.05	3.70	1.08	94.98	45.26
S10	30.01	15.40	0.34	0.16	0.33	0.02	0.16	0.10	0.24	0.07	0.18	0.05	0.17	0.05	3.84	1.15	123.80	62.85
S20	37.02	17.26	0.43	0.19	0.33	0.03	0.15	0.11	0.22	0.07	0.14	0.04	0.13	0.02	4.14	1.20	165.58	79.44
PB	14.25	9.79	0.16	0.10	**0.34**	0.04	**0.18**	0.11	0.29	0.05	**0.24**	0.04	**0.23**	0.03	3.02	0.98	48.06	37.82
SB	**12.56**	8.31	**0.15**	0.09	0.34	0.05	0.16	0.11	**0.30**	0.06	0.23	0.06	0.21	0.04	**2.78**	1.58	**28.36**	15.54
$t_{oa} = 0.25$																		
D8	13.69	8.81	0.16	0.10	0.30	0.04	0.14	0.11	0.24	0.07	0.13	0.02	0.13	0.02	3.43	1.68	56.15	34.77
S10	13.15	9.10	0.15	0.09	0.29	0.04	0.13	0.10	0.23	0.06	0.13	0.02	0.13	0.02	3.48	1.73	54.15	34.75
S20	18.43	9.65	0.22	0.11	0.30	0.04	0.12	0.10	0.21	0.08	0.13	0.02	0.13	0.02	3.85	1.86	85.93	45.28
PB	5.62	6.08	0.08	0.07	**0.29**	0.03	0.14	0.10	0.27	0.04	0.20	0.03	0.17	0.03	**3.07**	1.72	24.49	30.31
SB	**1.77**	1.44	**0.03**	0.02	**0.29**	0.03	**0.18**	0.11	**0.28**	0.03	**0.25**	0.05	**0.20**	0.06	4.60	2.61	**9.05**	6.15

All values are the average over the 84 different simulations. Bold values indicate the best performing algorithm for each metric with each threshold

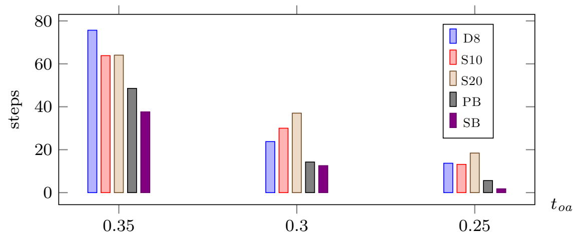


Fig. 10 Average number of steps (y-axis) required to find an optimal solution for each of the 3 tested thresholds (x-axis)

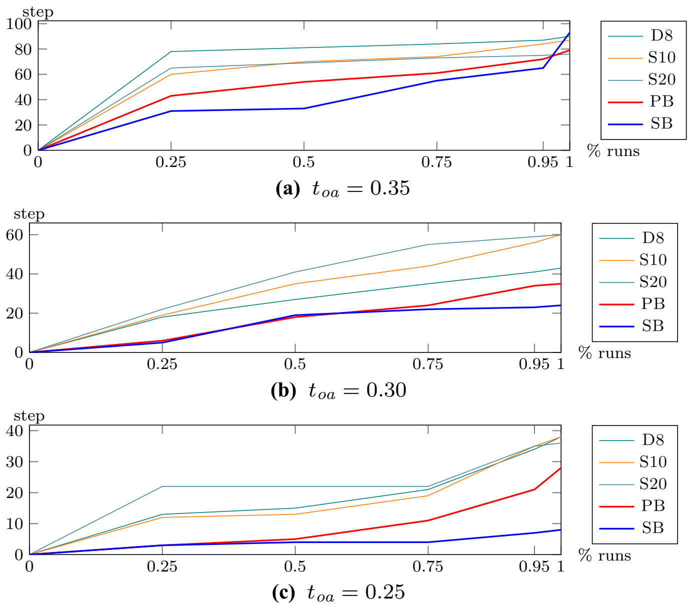


Fig. 11 The number of steps (y-axis) required by the tested heuristics to terminate the first (0.25), second (0.5), third (0.75) quartiles and the 95% (0.95 in the plot) of all the 84 executions (x-axis)

may however result less appropriate when bigger variations in the full set of requirements are needed. For instance, Fig.11 shows that, in case of $t_{oa} = 0.35$, SB could not find the optimal solution for two particular executions without trying almost all the possible system configurations (see SB in Fig. 11a between 0.95 and 1).

Configurations exploration In terms of the percentage of explored system configurations (reported in the *% explored config* column of Table 9), the results show that the informed strategies explore a smaller portion of the possible system configurations than the informed strategies. Notably, in case of $t_{oa} = 0.35$, SB results in a 39% improvement over the best uninformed strategy (S10). For $t_{oa} = 0.25$, such improvement increases to 78% over S10. In other terms, in order to find one of the 17.8% optimal solutions, SB needs to explore, on average, only about 3% of the system configurations.

Interpretation The results of our simulations *support* H_1: the informed heuristics converge quicker than uninformed heuristics, both in terms of number of steps and percentage of explored system configurations. This entails that probabilistic reasoning about monitored requirements seems to deliver an added value over the uninformed distance-based heuristics, in terms of convergence speed.

7.3.2 Quality (H_2)

Quality of the final solution All the tested algorithms stop searching for new system configurations when a system configuration that meets the desired threshold is identified. As such, the average objectives achievement joint probability of the final solution is always above the threshold t_{oa}. All tested heuristics provide on average similar results in terms of average objectives achievement joint probability of the final solution; the *Final conf* column of Table 9 reports such value. For $t_{oa} = 0.35$, all algorithms provide an average value of the final solution that is close to the overall best possible solution: the quality of the final solution is, on average, 0.49, which is 0.14 higher than the threshold. The average quality of the identified solutions decreases with the lower values for t_{oa}, and we do not see differences between the algorithms. This is an expected behavior, for hill climbing algorithms employ local search techniques that stop as soon as an optimal solution is found.

Quality throughout the process As described at the beginning of the section, we use different metrics for the analysis of the quality of the heuristics throughout the process of finding an optimal solution.

Figure 12 illustrates the trend of the average objectives achievement during the optimization process, until all algorithms terminated all the 84 executions. A value in the plots for an algorithm A at step i represents the average value (w.r.t. all the 84 executions) of the objectives achievement joint probability oa of the configurations tried at step i.

If we consider metric $\overline{oa}_A$ (i.e., the average quality of the solutions tried by an heuristic A before stopping, reported in column $\overline{oa}_A$ of Table 9), the best heuristic is PB, which in all cases provides on average better solutions throughout the process (see also the red line in Fig. 12. When we consider, instead, the initial phases of the optimization process (the first 5–6 steps in the three sub-figures), we see that SB outperforms PB, selecting higher-quality solutions. This, as seen in Sect. 7.3.1 in the case of $t_{oa} = 0.25$, allows to find a solution above the threshold in very few steps. Due to this behaviour, SB is always the fastest heuristics at reaching the desired threshold, outperforming the other strategies (and in particular the uninformed ones) also in terms of $\overline{oa}_{A,thres}$ (reported in column $\overline{oa}_{A,thres}$ of Table 9).

Figure 13 reports the average $avg_{i=1}^{n} p_{Ai}$, where $p_{Ai} = avg_{i=1}^{84} oa_{Ai}$ is the average objectives achievement joint probability for the configurations tried by algorithm A at simulation step i, and n is defined as follows:

- Figure 13a: the step when all algorithms terminate all executions: 92 for $t_{oa} = 0.35$, 59 for $t_{oa} = 0.30$, 37 for $t_{oa} = 0.25$.
- Figure 13b: the step when the first algorithm ends all its executions, i.e., 75 for $t_{oa} = 0.35$, 23 for $t_{oa} = 0.30$, and 7 for $t_{oa} = 0.25$.
- Figure 13c: the step when the first algorithm meets the threshold i.e., 34 for $t_{oa} = 0.35$, 19 for $t_{oa} = 0.30$, and 3 for $t_{oa} = 0.25$.

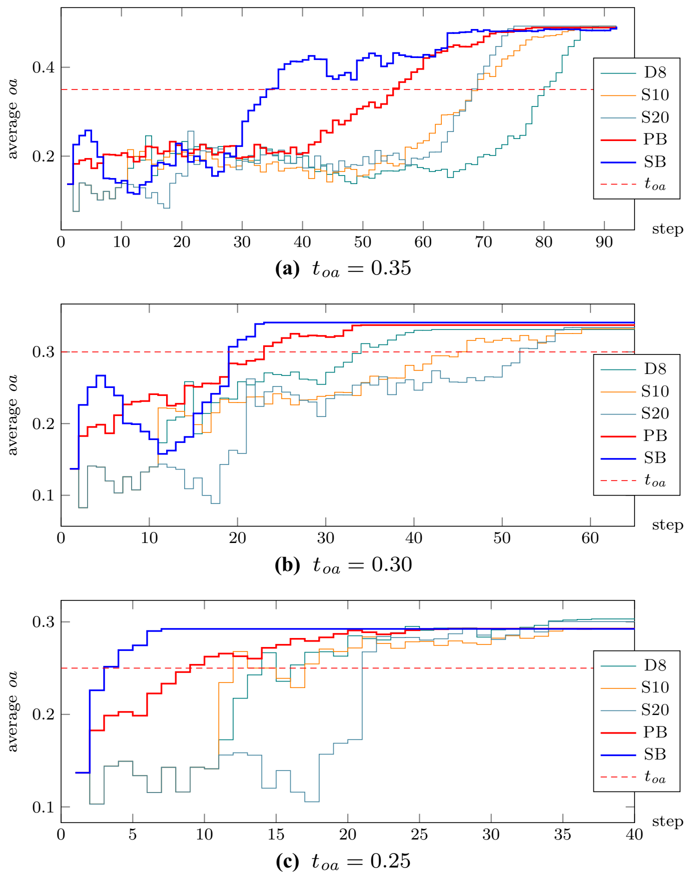
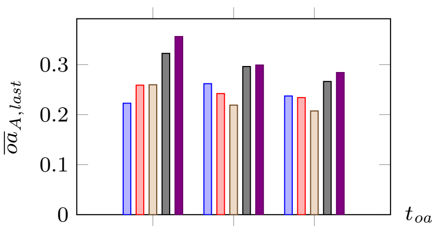


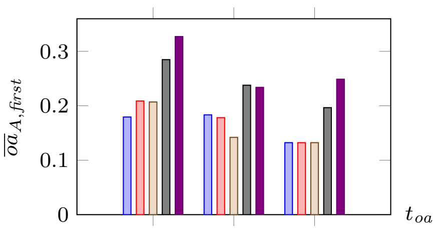
Fig. 12 The trend of the average objectives achievement joint probability (y-axis) of the solutions selected during the optimization process. The x-axis indicates the steps made by the hill climbing algorithm

The bar charts in Fig. 13 confirm the superiority of the informed algorithms when considering the average objective satisfaction rate throughout the process. The improvements are visible in all conditions, and become manifest if we consider the step when the first algorithm ends (metric $\overline{oa}_{A,first}$, illustrated in Fig. 13b and reported in column $\overline{oa}_{A,first}$ of Table 9): for $t_{oa} = 0.35$, the informed algorithms provide a gain of about 37–50% over the non-informed algorithms, and for $t_{oa} = 0.25$, the gain delivered by SB over the average of the uninformed algorithms is about 88%.

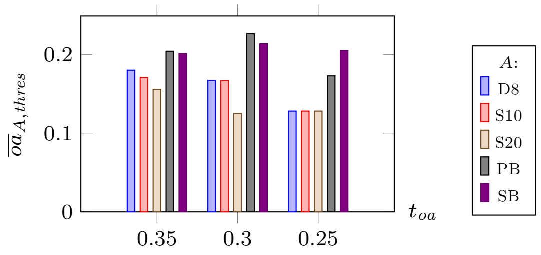
Concerning $\overline{oa}_{A,last}$ (Fig. 13a and column $\overline{oa}_{A,last}$ of Table 9), since SB terminates the majority of the solutions early in the optimization process, its value of $\overline{oa}_{A,last}$ (which includes in the average also the oa of the executions already terminated) is generally higher than other metrics. However, due to the overall high quality of the solutions explored by PB, in some cases its performance is slightly better than SB. In general, Fig. 12 highlights that PB exhibits a more stable behavior in terms of quality of explored system configuration, if compared to SB.



(a) When all algorithms end



(b) When the first algorithm ends



(c) When the first algorithm meets the threshold

Fig. 13 Average objectives achievement joint probability (y-axis) throughout the process of finding an optimal solution, for all the 3 tested thresholds (x-axis)

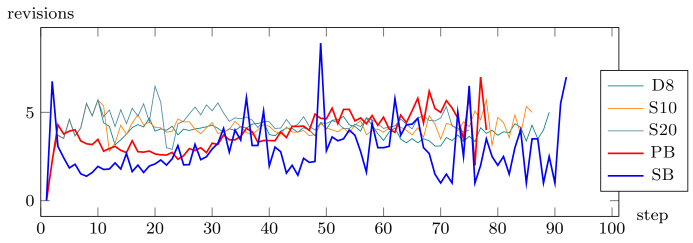
Interpretation The results of our simulations *partially support* H_2: while the quality of the final solution is not affected greatly by the algorithm, the informed heuristics show a higher objectives achievement joint probability throughout the process of finding an optimal system configuration.

7.3.3 Stability (H_3)

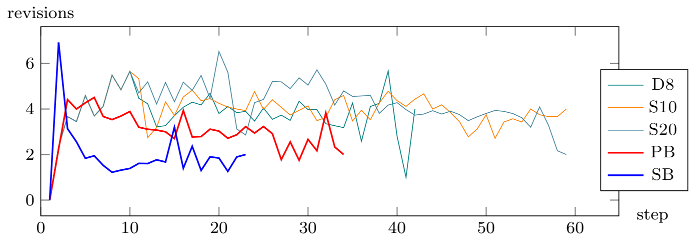
As reported in Sect. 7.3.2, despite SB and PB are comparable when it comes to the average objectives achievement joint probability, PB exhibits a more consistent behavior than SB, which instead leads to more intense oscillations, due to the more heterogeneous definition of the neighborhood.

To better understand these differences and similarities, we focus on *stability metrics* in terms of the number of performed requirement revisions. This metric matters, for each revision may incur some costs (e.g., to deploy the necessary sensors for monitoring requirements compliance), which the system designer wants to minimize. Figure 14 reports the trend of the average number of revision performed at each step of the optimization process. Note that the lines for an individual algorithm end when all the 84 executions of that algorithm terminate. The figure reports, at a given step i, the average number of revisions performed w.r.t. the executions that are still running at step i. Conversely, the executions that already terminated are not considered when computing the average. An approximation of the number of executions still running at a certain step i can be seen in Fig. 11.

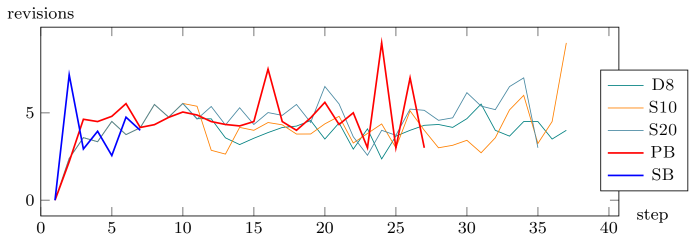
The total number of revisions for SB to find an optimal solution is lower than the other algorithms in all cases (see the *Total revs* column of Table 9). This is particularly



(a) $t_{oa} = 0.35$



(b) $t_{oa} = 0.30$



(c) $t_{oa} = 0.25$

Fig. 14 Average number of revisions (y-axis) performed at each step (x-axis) of the optimization process

evident with the lowest threshold $t_{oa} = 0.25$, since SB finds a solution in very few steps, leading to an average total number of revisions of 9.05.

Concerning the average number of revisions performed at each step (column *Revisions per step* in Table 9), note that all algorithms are comparable with thresholds $t_{oa} = 0.35$ and $t_{oa} = 0.3$, which require more steps than $t_{oa} = 0.25$. However, in the initial phases of the optimization process (first steps), SB performs an higher number of revisions per step, compared to the other algorithms. This explains why, with $t_{oa} = 0.25$, it reaches an optimal solution faster than the others.

Interpretation The results of our simulations *support* H_3: the total number of revisions that informed heuristics make is lower than the number for uninformed heuristics. However, to do so, the informed algorithms make—in some cases—more revisions per step than the uninformed heuristics.

7.4 Noise tolerance for the informed algorithms (H_4)

The performance of the informed heuristics depends on the quality of the data analysed. So far, we assumed our system can monitor the satisfaction of the requirements

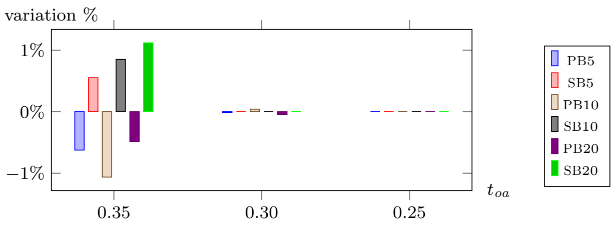


Fig. 15 The variation of the percentage of system configurations explored (y-axis) when introducing 5%, 10% and 20% of noise in the input data (respectively identified with PB5 and SB5, PB10 and SB10 and PB20 and SB20), for each of the 3 tested thresholds (x-axis)

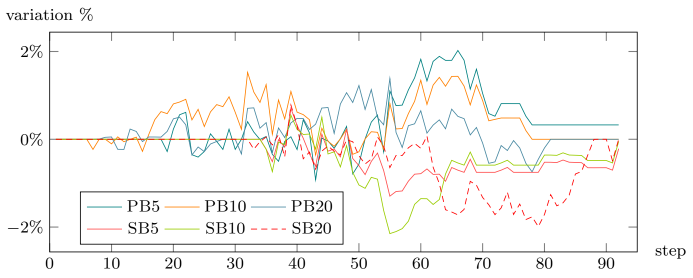


Fig. 16 The trend of the variation of the average objectives achievement joint probability (y-axis) over the steps of the optimization process (x-axis) in case of $t_{oa} = 0.35$ when noise is introduced

perfectly. We now relax this assumption to analyze how the algorithms perform in the presence of noisy data about requirements satisfaction.

We compare the performance when a certain percentage p of the information acquired from the *Monitoring* component is incorrect. In particular we analyze results with $p = 5\%$, 10%, and 20%. To do so, we modified our dataset by uniformly altering $p\%$ of the data concerning active requirements satisfaction. Specifically, we changed, with probability p, every value *obeyed* and *violated* in Table 3, respectively into *violated* and *obeyed*.

Figure 15 reports the results for the variation of the percentage of explored system configurations by the two informed heuristics PB and SB when introducing noise. Even with 20% of noise, with $t_{oa} = 0.30$ and $t_{oa} = 0.25$, the algorithms presents almost no difference in the system configurations selected. In case of $t_{oa} = 0.35$, when more system configurations need to be explored, the maximum detected variation is of about the 1% in terms of system configurations explored, when perturbing 20% of the input data.

Figure 16 shows the impact of noise on the average objectives achievement joint probability over the steps of the optimization process in case of $t_{oa} = 0.35$, the only threshold level at which the introduced noise had some noticeable impact. The figure shows no impact during the early phases of the optimization process, while the effect is visible after several steps of optimization, due to the presence of an increasing quantity of noisy data. The effects, however, are within the 2% range. The line chart also helps understand why the algorithms are not impacted with lower thresholds: the effect of noise occurs after multiple steps, while our algorithms return before such effects are visible.

Interpretation The results of our simulations *support* H_4: the informed heuristics seem to have high tolerance to degrees of noise up to 20%. However, it must be noted that data was perturbated in a uniform manner, and this may have positively affected the ability to tolerate noise.

7.5 Threats to validity

The implementation of the prototype of the control loop described in Sect. 6.2 could be incorrect, which would render the results invalid. We reduced the potential impact of this threat by performing an extensive testing of the implementation and by applying it to different problems.

The chosen topology of the *Requirement Bayesian Network*, reflecting the structure of a requirements model, may influence the conclusions drawn via probabilistic inference. The choice of such topology is a threat to construct validity. For example, we do not capture causal relationships between sibling requirements or between objectives, which may help better explain when and why the requirements and the objectives are achieved. Different mappings between a requirements model may be tried to overcome this limitation.

The interpretation of the results is subject to the size and type of the set of system configurations tested. To mitigate this threat, we paid attention to our interpretation and the wording of the implications, and we deliberately omitted tests for statistical significance due to the use of a single case.

The notion of degree of validity is based on the assumption that the collected positive and negative evidence have the same statistical significance. The choice of such method to evaluate the assumptions affects construct validity. Additional probabilistic learning techniques should be explored and tested.

Finally, our conclusions have only limited generalizability. It is possible that the proposed algorithms behaves differently on different problems. This threat to conclusion validity is partly mitigated by our previous work (Dell'Anna et al. 2018b), where we applied the same approach to a different problem, obtaining analogous results.

8 Related work

The intrinsic dynamism of modern software systems leads to high runtime uncertainty (Whittle et al. 2010; Lehman 2005), which makes adaptation a necessity. Researchers argue for the necessity of evaluating the assumptions made during the design of a system in order to support software evolution. In their seminal works (Lehman and Ramil 2003; Lehman 2005), Lehman et al., identify invalid assumptions as one the main causes for software evolution. They highlight how the—implicit or explicit—presence of assumptions in (E-type) software is inevitable and follows from the fact that real-world software and the environment in which it operates have a potentially unbounded number of properties.

Several approaches for the evaluation of assumptions have been proposed over the years. Boness et al. (2008, 2011) explicitly represent assumptions when defining requirements within a goal oriented framework. They use such concept to help the system designers to assess, during requirement analysis, the confidence in (and the risk due to) the set of elicited requirements. In order to do so they integrate expert knowledge, argumentation techniques and propagation of confidence values through the goal graph. In our work, we consider assumptions that are implicit in the structure of a requirements model (rather than explicitly represented as in Boness et al. 2011) and we calculate (and make use of) their degree of validity at runtime by means of probabilistic reasoning on a Bayesian Network trained with data obtained during the execution.

The availability of a requirements model during execution (Blair et al. 2009; Bencomo et al. 2010) is crucial to build a framework that supports the runtime evolution of the system requirements. Several frameworks exist that use such models to support the monitoring and diagnosis of requirements (Fickas and Feather 1995; Wang et al. 2009; Robinson 2006). Such approaches are powerful and allow the identification of deviations from the requirements. However, they do not challenge the validity of the requirements (models) themselves.

Ali et al. (2011) illustrate the advantages of monitoring requirements at runtime to detect when design-time assumptions concerning requirements satisfaction become invalid. They also discuss the importance of keeping track of the relationship between context and requirements at runtime (Ali et al. 2013). Paucar et al. (2017) propose techniques to reassess the assumptions about the priority of non-functional requirements.

Models at runtime are often used for guiding the adaptation of the system. Souza et al. (2011) define awareness requirements as meta-requirements to drive adaptations. Non-functional requirements (NFRs) have been used to trigger and guide self-adaptation; for example, contributions to objectives can help identify those system configurations that maximize NFR satisfaction (Salehie and Tahvildari 2012; Dalpiaz et al. 2013).

These approaches constitute our baseline: our framework uses requirements models at runtime (Blair et al. 2009; Bencomo et al. 2010), can rely on existing monitoring frameworks (Wang et al. 2009; Robinson 2006), and implements the idea of reconsidering design assumptions at runtime (Ali et al. 2011). The distinguishing features of our approach are the focus on sociotechnical systems, the use of Bayesian learning, and the employment of an hill climbing approach to identify an optimal system configuration.

In order to support the automated requirements evolution, Whittle et al. (2010) propose the notion of requirements revision. They present a requirements language for self-adaptive systems (RELAX) that allows to specify relaxed versions of a requirement during the elicitation phase. Existing requirements revision approaches mainly focus on re-assessing the weights of non-functional requirements (Almeida et al. 2015; Bencomo 2015). Knauss et al. (2016) discuss the mining of optimal contexts for previously defined contextual requirements, and propose a revision of the contextual condition of applicability of the requirements.

The normative multiagent systems (NorMAS) literature offers techniques for the dynamic update of norms that regulate a multiagent systems. Aucher et al. (2009) introduce a dynamic context logic that describes the operations of contraction and expansion of theories by introducing or removing new rules. Governatori and Rotolo (2010) investigate the legal consequences of applying theory revision to reason about legal abrogations and annulments. Alechina et al. (2014) show how to formally obtain an approximated version of a norm to cope with imperfect monitors for the original norm. Since norms are an important type of requirements for STSs (Singh 2013; Chopra et al. 2014), NorMAS research is a rich cross-fertilization tool for the RE discipline.

In previous work (Dell'Anna et al. 2018a), we proposed Bayesian Networks as a tool to learn, from runtime data, the correlation between the satisfaction of requirements and the achievement of the overall system objectives in different operating contexts. In Dell'Anna et al. (2018a) we show that such information can be used to validate some assumptions made in a goal model. In this paper, we embedded our requirements assumptions validation technique within a holistic framework for the evolution of STSs; in particular, the validity of the assumption is used by our heuristic algorithms that perform runtime requirements revision.

Cailliau and van Lamsweerde (2013) present a technique for the quantitative assessment of requirements-related risks. Their framework uses KAOS goal models extended with a probabilistic layer to evaluate the consequences of explicit obstacles on the satisfaction of goals. The concept of obstacles is similar to the concept of harmful requirements presented in this paper; however, we do not consider requirements that are known to be harmful *a priori*. We focus, instead, on techniques for discovering at runtime whether some requirements are useful or not and in which contexts.

Our requirements revision types (relaxation, strengthening, disabling, etc.) are similar to the strategies presented by van Lamsweerde et al. (1998) to resolve goal conflicts. In their work, such strategies are applied in the early stages of requirements elicitation (at design time) and they rely on the available domain knowledge. Our framework focuses on the runtime analysis of the requirements and of the assumptions made at design time. A deeper study of the relationship of our work with runtime conflict resolution techniques is left for future work.

9 Discussion and future work

We introduced a novel framework for guiding the evolution of sociotechnical systems. Our approach uses requirements models to represent system objectives, requirements, and their relationships. The framework supports both the *manual evolution* of the STS, by revealing the validity of the assumptions in a requirements model, and the *automated adaptation*, by revising the requirements in order to quickly identify an optimal system configuration.

This work employs two techniques from artificial intelligence: (i) *Bayesian Networks* as a tool to learn and reason about the relationship between contexts, requirements and objectives based on evidence from system execution; and (ii) *hill climbing algorithms* as a technique to explore the space of alternative configurations of the STS and identify optimal configurations that maximize the satisfaction of the objectives.

Our experiments with a smart traffic simulator show promising results for our automated requirements revision algorithms. Both the PB and SB heuristics that we propose outperform uninformed hill climbing heuristics. The requirement revisions are guided, in our algorithms, by the information retrieved from runtime execution data about the validity of the assumptions made in a requirements model. The results show that using that information allows to accelerate convergence to an optimal configuration by guiding the requirement revision process. Our heuristics provided, in certain cases (see Sect. 7.3.1), an efficiency gain of about 75% compared to uninformed heuristics, in terms of number of the explored configurations of requirements. The heuristic SB was able to terminate 95% of all its executions before all the baseline uninformed heuristics could reach their first quartile. In one experimental setting, SB was able to find, on average, an optimal configuration in less than 2 steps, exploring about 3% of possible configurations of requirements in order to find one of the 17.8% optimal ones.

The results revealed that our informed algorithms positively affect the quality of the attempted system configurations. When considering the average stakeholders' objective satisfaction rate throughout the optimization process, our informed heuristics provided an improvement, compared to uninformed ones, ranging from 37 to 88% (see Sect. 7.3.2 for more details).

Our analysis of the stability of the algorithms, in terms of number of revisions, showed also that our informed heuristic SB suits well those problems for which an optimal solution needs to be found quickly with a small total number of revisions along the process. Compared to the other tested algorithms, however, SB includes steps in which it performs a high absolute number of revisions (see, for instance, the initial peaks in Fig. 14). Should there be a limit on the maximum number of acceptable requirement revisions, other heuristics could be preferable. A possible reason why this factor may matter is that revising requirements may pose some challenges for humans to adapt to the new requirements (e.g., think of revising the speed limits of all streets at the same time).

Finally, our proposed algorithms exhibited a high tolerance to possible noise in the data used to train the Bayesian Network: a uniform perturbation of 20% of the input data by introducing erroneous information about requirements satisfaction lead only to a variation of about 1% in terms of number of system's configurations explored during the optimization process and impacted less than 2% on the average stakeholders' objectives satisfaction.

Limitations and future work A thorough evaluation of the scalability, usefulness and generality of our proposal is imperative. So far, we have focused on smart traffic simulations because this is an example of an STS that has numerous simulator frameworks. Our current Bayesian Network assumes a consistent behavior of the STS population over time. *Dynamic Bayesian Networks* (Russell and Norvig 2010) should be considered to support more dynamic STSs, in which we cannot make such assumption. Furthermore, the two revision algorithms that we introduced do not store any information concerning the effects of the requirement revisions applied. Refined revision algorithms shall be developed with a larger memory than just the current configuration; possible techniques include Q-Learning (Rummery and Niranjan 1994) and Dynamic Decision Networks (Russell and Norvig 2010). Moreover, we plan to develop algorithms that can guide software evolution by providing additional information on the most critical and significant assumptions. To do so, we plan to employ other analysis techniques for Bayesian Networks, such as sensitivity analysis (van der Gaag et al. 2007) or qualitative reasoning (Wellman 1990). Visualization tools, which are missing in this paper, are necessary to support human analysts in visualizing the validity of the assumptions in a requirements model, as discussed in Reddivari et al. (2014), and to help them in deciding about the manual evolution of an STS. A starting point could be the visualization we developed in earlier work (Dell'Anna et al. 2018a). Finally, while this work focuses on the revision of requirements by exploring the space of alternatives within a model, it would be interesting also to explore the possibility to synthesize new requirements that are not included in the given model.

References

Alechina, N., Dastani, M., Logan, B.: Norm approximation for imperfect monitors. In: Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, pp. 117–124 (2014)
Ali, R., Dalpiaz, F., Giorgini, P., Souza, V.E.S.: Requirements evolution: from assumptions to reality. In: Proceedings of the EMMSAD, pp. 372–382 (2011). https://doi.org/10.1007/978-3-642-21759-3_27
Ali, R., Dalpiaz, F., Giorgini, P.: Reasoning with contextual requirements: detecting inconsistency and conflicts. Inf. Softw. Technol. **55**(1), 35–57 (2013). https://doi.org/10.1016/j.infsof.2012.06.013
Almeida, A., Bencomo, N., Batista, T.V., Cavalcante, E., Dantas, F.: Dynamic decision-making based on NFR for managing software variability and configuration selection. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1376–1382 (2015). https://doi.org/10.1145/2695664.2695875
Aucher, G., Grossi, D., Herzig, A., Lorini, E.: Dynamic context logic. In: Proceedings of the 2nd International Workshop on Logic, Rationality, and Interaction, pp. 15–26 (2009)
Bencomo, N.: Quantun: Quantification of uncertainty for the reassessment of requirements. In: Proceedings of the 23rd IEEE International Requirements Engineering Conference, pp. 236–240 (2015). https://doi.org/10.1109/RE.2015.7320429
Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: requirements as runtime entities. Proc. ICSE **2**, 199–202 (2010)
Bencomo, N., Belaggoun, A., Issarny, V.: Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: Proceedings of the SEAMS, pp. 113–122 (2013). https://doi.org/10.1109/SEAMS.2013.6595498
Blair, G., Bencomo, N., France, R.B.: Models@ run. time. Computer **42**(10), 22–27 (2009)
Boness, K., Finkelstein, A., Harrison, R.: A lightweight technique for assessing risks in requirements analysis. IET Softw. **2**(1), 46–57 (2008). https://doi.org/10.1049/iet-sen:20070068
Boness, K., Finkelstein, A., Harrison, R.: A method for assessing confidence in requirements analysis. Inf. Softw. Technol. **53**(10), 1084–1096 (2011). https://doi.org/10.1016/j.infsof.2011.05.003
Cailliau, A., van Lamsweerde, A.: Assessing requirements-related risks through probabilistic goals and obstacles. Requir. Eng. **18**(2), 129–146 (2013). https://doi.org/10.1007/s00766-013-0168-5
Chopra, A.K., Dalpiaz, F., Aydemir, F.B., Giorgini, P., Mylopoulos, J., Singh, M.P.: Protos: Foundations for engineering innovative sociotechnical systems. In: Proceedings of the 22nd IEEE International Requirements Engineering Conference, pp. 53–62 (2014)
Cziharz, T., Hruschka, P., Queins, S., Weyer, T.: Handbook of requirements modeling according to the IREB standard. Handbook, International Requirements Engineering Board (2016)
Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Adaptive socio-technical systems: a requirements-based approach. Requir. Eng. **18**(1), 1–24 (2013). https://doi.org/10.1007/s00766-011-0132-1

Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 language guide. arXiv:1605.07767 [cs.SE] (2016)

De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems II, pp. 1–32. Springer, Berlin (2013)

Dell'Anna, D., Dalpiaz, F., Dastani, M.: Validating goal models via Bayesian networks. In: Proceedings of the International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'18) (2018a)

Dell'Anna, D., Dastani, M., Dalpiaz, F.: Runtime norm revision using Bayesian networks. In: Proceedings of the 21st International Conference on Principles and Practice of Multi-Agent Systems (2018b)

Doguc, O., Ramirez-Marquez, J.E.: A generic method for estimating system reliability using Bayesian networks. Rel. Eng. Syst. Saf. **94**(2), 542–550 (2009). https://doi.org/10.1016/j.ress.2008.06.009

Fenton, N.E., Neil, M., Marsh, W., Hearty, P.S., Marquez, D., Krause, P., Mishra, R.: Predicting software defects in varying development lifecycles using Bayesian nets. Inf. Softw. Technol. **49**(1), 32–43 (2007). https://doi.org/10.1016/j.infsof.2006.09.001

Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Second IEEE International Symposium on Requirements Engineering, March 27–29, 1995, York, England, pp. 140–147 (1995). https://doi.org/10.1109/ISRE.1995.512555

Filieri, A., Ghezzi, C., Tamburrelli, G.: A formal approach to adaptive software: continuous assurance of non-functional requirements. Form. Asp. Comput. **24**(2), 163–186 (2012). https://doi.org/10.1007/s00165-011-0207-2

Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Proceedings of the 21st International Conference on Conceptual Modeling, Vol. 2, pp. 167–181 (2002)

Governatori, G., Rotolo, A.: Changing legal systems: legal abrogations and annulments in defeasible logic. Logic J. IGPL **18**(1), 157–194 (2010). https://doi.org/10.1093/jigpal/jzp075

Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H.A., Thomo, A.: Acon: a learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf. Softw. Technol. **70**, 85–99 (2016). https://doi.org/10.1016/j.infsof.2015.10.001

Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A survey on engineering approaches for self-adaptive systems. Pervasive Mob. Comput. **17**, 184–206 (2015)

Lehman, M.M.: The role and impact of assumptions in software development, maintenance and evolution. In: IEEE International Workshop on Software Evolvability (Software-Evolvability'05), pp. 3–14. IEEE (2005)

Lehman, M.M., Ramil, J.F.: Software evolution—background, theory, practice. Inf. Process. Lett. **88**(1–2), 33–44 (2003). https://doi.org/10.1016/S0020-0190(03)00382-X

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: a multiagent simulation environment. Simulation **81**(7), 517–527 (2005)

Mendes, E., Mosley, N.: Bayesian network models for web effort prediction: a comparative study. IEEE Trans. Softw. Eng. **34**(6), 723–737 (2008). https://doi.org/10.1109/TSE.2008.64

Misirli, A.T., Bener, A.B.: A mapping study on Bayesian networks for software quality prediction. In: 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2014, Hyderabad, India, June 3, 2014, pp. 7–11 (2014). https://doi.org/10.1145/2593801.2593803

Paucar, L.H.G., Bencomo, N., Yuen, K.K.F.: Juggling preferences in a world of uncertainty. In: Proceedings of the RE, pp. 430–435 (2017)

Reddivari, S., Rad, S., Bhowmik, T., Cain, N., Niu, N.: Visual requirements analytics: a framework and case study. Requir. Eng. **19**(3), 257–279 (2014). https://doi.org/10.1007/s00766-013-0194-3

Robinson, W.N.: A requirements monitoring framework for enterprise systems. Requir. Eng. **11**(1), 17–41 (2006). https://doi.org/10.1007/s00766-005-0016-3

Rummery, G.A., Niranjan, M.: On-Line Q-Learning Using Connectionist Systems, vol. 37. University of Cambridge, Cambridge (1994)

Russell, S.J., Norvig, P.: Artificial Intelligence—A Modern Approach, 3, internat edn. Pearson Education, London (2010)

Salehie, M., Tahvildari, L.: Towards a goal-driven approach to action selection in self-adaptive software. Softw. Pract. Exp. **42**(2), 211–233 (2012). https://doi.org/10.1002/spe.1066

Schmid, S., Gerostathopoulos, I., Prehofer, C., Bures, T.: Self-adaptation based on big data analytics: a model problem and tool. In: Proceedings of SEAMS, pp. 102–108 (2017). https://doi.org/10.1109/SEAMS.2017.20

Scutari, M.: Learning Bayesian networks with the bnlearn R package. arXiv preprint arXiv:0908.3817 (2009)
Singh, M.P.: Norms as a basis for governing sociotechnical systems. ACM Trans. Intell. Syst. Technol. **5**(1), 21:1–21:23 (2013). https://doi.org/10.1145/2542182.2542203
Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M.Z., McDermid, J.A., Paige, R.F.: Large-scale complex IT systems. Commun. ACM **55**(7), 71–77 (2012). https://doi.org/10.1145/2209249.2209268
Souza, V.E.S., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems. In: Proceedings of the SEAMS, pp. 60–69 (2011). https://doi.org/10.1145/1988008.1988018
Spiegelhalter, D.J., Dawid, A.P., Lauritzen, S.L., Cowell, R.G.: Bayesian analysis in expert systems. Stat. Sci. **8**, 219–247 (1993)
Tsvetovat, M., Carley, K.M.: Modeling complex socio-technical systems using multi-agent simulation methods. KI **18**(2), 23–28 (2004)
van der Gaag, L., Renooij, S., Coupé, V.: Sensitivity analysis of probabilistic networks. Adv. Probab. Graph. Models **214**, 103–124 (2007)
Van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software, vol. 10. Wiley, Chichester (2009)
van Lamsweerde, A., Darimont, R., Letier, E.: Managing conflicts in goal-driven requirements engineering. IEEE Trans. Softw. Eng. **24**(11), 908–926 (1998). https://doi.org/10.1109/32.730542
Wang, Y., McIlraith, S.A., Yu, Y., Mylopoulos, J.: Monitoring and diagnosing software requirements. Autom. Softw. Eng. **16**(1), 3–35 (2009). https://doi.org/10.1007/s10515-008-0042-8
Wellman, M.P.: Fundamental concepts of qualitative probabilistic networks. Artif. Intell. **44**(3), 257–303 (1990). https://doi.org/10.1016/0004-3702(90)90026-V
Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.: RELAX: a language to address uncertainty in self-adaptive systems requirement. Requir. Eng. **15**(2), 177–196 (2010). https://doi.org/10.1007/s00766-010-0101-0
Wu, P.P.Y., Fookes, C., Pitchforth, J., Mengersen, K.: A framework for model integration and holistic modelling of socio-technical systems. Decis. Support Syst. **71**, 14–27 (2015)
Yu, E., Mylopoulos, J.: Why goal-oriented requirements engineering. In: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, vol. 15, pp. 15–22 (1998)