Hindawi Publishing Corporation
Advances in Artificial Intelligence
Volume 2016, Article ID 2959508, 15 pages
http://dx.doi.org/10.1155/2016/2959508

Research Article

Weighted Constraint Satisfaction for Smart Home Automation and Optimization

Noel Nuo Wi Tay, János Botzheim, and Naoyuki Kubota

Graduate School of System Design, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan

Correspondence should be addressed to Noel Nuo Wi Tay; tay-noelnuowi@ed.tmu.ac.jp

Received 7 April 2016; Accepted 5 October 2016

Academic Editor: Roman Bartak

Automation of the smart home binds together services of hardware and software to provide support for its human inhabitants. The rise of web technologies offers applicable concepts and technologies for service composition that can be exploited for automated planning of the smart home, which can be further enhanced by implementation based on service oriented architecture (SOA). SOA supports loose coupling and late binding of devices, enabling a more declarative approach in defining services and simplifying home configurations. One such declarative approach is to represent and solve automated planning through constraint satisfaction problem (CSP), which has the advantage of handling larger domains of home states. But CSP uses hard constraints and thus cannot perform optimization and handle contradictory goals and partial goal fulfillment, which are practical issues smart environments will face if humans are involved. This paper extends this approach to Weighted Constraint Satisfaction Problem (WCSP). Branch and bound depth first search is used, where its lower bound is estimated by bacterial memetic algorithm (BMA) on a relaxed version of the original optimization problem. Experiments up to 16-step planning of home services demonstrate the applicability and practicality of the approach, with the inclusion of local search for trivial service combinations in BMA that produces performance enhancements. Besides, this work aims to set the groundwork for further research in the field.

1. Introduction

As human lifestyle has become increasingly hectic, smart home starts to gain more prominence. Smart home is a viable solution to provide comfort for its human inhabitants, monitoring, and caregiving for the elderly. Demographic shifts towards elderly people for developed and some developing countries are a serious issue, where caregivers are in short supply relative to the population of the elderly, which is a serious issue in Japan as the elderly population is expected to reach 25.2% of the total population. Many of them will lose the ability to live independently [1, 2]. With the rising number of the elderly coupled by hectic lifestyle of the working class, the elderly cannot be provided with ample care that they need. Therefore, it is necessary for the home to provide support to handle mundane tasks and to ensure the continual welfare of its human inhabitants.

Smart homes need to provide services based on the devices that they are connected to. Given the available devices and initial state of the house, smart homes should coordinate their services in order to reach the maximum number of goals set from either human command or event triggered by generating an optimal sequence of plans. An optimum plan can mean a plan that minimizes energy consumption or maximizes comfort. For this paper, optimization is defined as minimizing a certain quantifiable cost. Besides that, the plan should also abide by constraints such as user preferences and physical limitations. An example of constraint related to user preference is the need for a room to be well lit when the user is in there while awake. An example for physical limitation constraint can be that the curtain needs to be open before the window can be swung open.

Researches related to web services and semantic web provide concepts and technologies that can support service discovery, the design and implementation of services, and providing machine understandable semantics for reasoning. The idea is adopted by ubiquitous and pervasive computing due to its ability to discover services and support for late binding of services.

Advancements in the web of things and web service technologies can shed light on service composition for building automation of the smart home. Similarities are quite significant, where both are dealing with high heterogeneity of objects, dynamic environment due to objects connection and disconnection from the system or due to contextual changes, and the need for integration and coordination of different objects to deliver complex services. Service oriented architecture (SOA) [3] complemented by automatic service composition provides the crucial approach in delivering what is required of home automation. SOA consists of independent but interoperable services that are loosely coupled, where each service only exposes its functionality to others, keeping the details of implementation hidden.

As argued by [4, 5], the main focus of previous platforms of pervasive applications does not consider complex and intelligent functionalities involving higher levels of information. Dynamic composition of complex sequence of services under uncertainty is necessary. Currently, to enable higher functionalities, tedious procedural programming is required from the user side. Besides, certain applications require manipulation and reading of values, such as counting the number of times a person goes to the toilet. Reference [4] uses artificial intelligence method, namely, constraint programming, to handle such service composition task where variables of a wider domain are required in the planning process. It performs service composition through solving constraint satisfaction problem (CSP) and implements individual services by appropriate devices. Hard constraints are used, and therefore the approach cannot (or at least very inefficiently) support optimization and partial goal fulfillment. Partial fulfillment of goals is important because certain goals might contradict each other or because they might be too complicated to solve in one go. For this, solving via hard constraints will conclude that the goal is unsatisfiable instead of trying to fulfill as many goals as possible.

For example, a given mother is requesting the TV to switch to channel 1, whereas her daughter requests channel 2. These two goals contradict each other. Service composition via hard constraints will conclude that the goals are not satisfiable, instead of trying to fulfill one of their wishes (such as switching to channel 1 to fulfill the mother's wishes as the goal imposed by her is considered much more important compared to the daughter). Such problems can be solved with soft constraints by introducing weights.

This work is built on the previous work on Informationally Structured Space (ISS) [6] by providing it with automated planning and optimization. It aims to extend the smart home automated planning approach in [4] from using CSP to Weighted Constraint Satisfaction Problem (WCSP). WCSP endows constraints with weights, which transforms them into soft constraints where optimization can be implemented. Branch and bound with depth first search is applied to solve the planning problem represented as WCSP, where the lower bound is estimated via bacterial memetic algorithm (BMA). Memetic algorithms have been shown to be effective in handling weighted constraints that give good enough solutions [7]. Test on various plan sequences and modifications shows the applicability of the approach in generating optimal plans in fulfilling the maximum number of goals. It lays the necessary groundwork and obtains implications for further development. This work will not cover the middleware implementation. Services are described by preconditions, effects, and properties, which resemble those of web services, such that they can be easily translated to current web service technologies.

This work is on high level planning. We like to consider smart devices such as the Google Nest that is capable of learning and adjusting as a composed service that we can call upon, or there are inferred results, which can be used by planning. This is likened to the automatic setup of surveillance system or robot navigation to a certain cabinet in a certain room. The former can be achieved through service composition via association with SWRL or SPARQL language [8]. The latter can have a built-in path finding algorithm that works with high level planning, where high level planning has to ensure proper working environment for low-level functions [9]. Theoretically, planning itself can handle all the tasks, including device association and low-level path finding. But, due to efficiency issue, it is considered that composed service and low-level tasks can be called upon as an actor of a particular planning operator.

The outline of the paper is as follows. Section 2 discusses the related works on home automation and service composition. The approach to solve planning problem through WCSP is presented in Section 3. Experiments are provided in Section 4 to demonstrate the applicability of the approach. Finally, the work is concluded in Section 5.

2. Background

The main focus of this paper is on home automation. Various home automation systems have been developed over time that employ the SOA architecture complemented by web technologies.

Home automation for inferring environment state is developed in [10]. It uses DogOnt ontology [11] to determine environment states like whether a certain room is smoke-free and mosquito-free depending on which rooms are adjacent and whether certain doors or windows are open or not. This work shows that additional information of environments can be inferred by defining a set of rules. Although it does not include service composition, since it only deals with sensors, nevertheless, it provides insight into generating rules as goals for service composition to work on.

The work in [12] uses knowledge representation and automated reasoning to create a self-adapting framework for managing user profiles and device services. Rule matching need not be exact, where it allows potential or intersection matches, which is the case in most practical situations. It supports partial or disjoint matches, where requests and supplied resources have conflicting features and when no other matches are possible. Service composition is performed through constant running of concept abduction algorithm [13] and selecting services to cover missing features. But such service composition will not be applicable if complex sequence of service is needed. Situation is made worse under uncertainty. A more flexible system is developed in [14],

where the case study is to maximize human comfort and energy efficiency.

Building automation adopting SOA and device profile for web service (DPWS) is proposed in [5]. Context information is obtained for processing before being used to guide service composition, complemented by policy rules and composition plans. To handle dynamic changes during service composition, it uses service composition plans to describe users requirements by Composition Plan Description Language (CPDL). Similar to [12], service composition does not support construction of complex sequence of services, such as the case of robot query on users preferred activity before preparing the environment for such activity.

Kaldeli et al. [4, 15] developed a home automation service composition based on SOA that is capable of complex composition of a sequence of services using constraint programming. Unlike previous methods, the system does not require manual construction of subplans. It handles context awareness and is able to deal with an uncertain situation by dynamic replanning. By using constraint programming, it can also handle variables with large domain efficiently. But the work employs hard constraints. It cannot fulfill partial goals as well as optimizing plans.

Home automation requires service composition to be functional. Here, we will review some service composition methods that are applied or are closely overlapping with the methods used in home automation. More detailed review on service composition can be obtained from [16].

In [17], OWLS-XPlan uses the semantic descriptions of atomic web services defined in OWL-S for planning purposes. Given the atomic services, the XPlan planning module will generate the sequence of services to fulfill a goal. In the work, an XML dialect of planning domain definition language (PDDL) is developed; thus, the system is PDDL compliant. Although the system obtains semantic descriptions in OWL-S, it is not utilized and semantic awareness is not achieved. Exact matching for service inputs and outputs is required from the planning module. This is the downside, as the most practical situation does not allow exact matching.

A framework developed in [18, 19] converts web service composition tasks into planning problems expressed in PDDL. The framework will then convert the devised plan into an OWL-S composite process description. The framework translates atomic OWL-S processes to planning operators, from which it derives the set of actions to achieve goals. Contrary to [17], when no exact composite services can be found, semantic information is utilized to obtain composite services that best approximate the goal.

To deal with complex tasks and to reduce planning complexity, hierarchical task network (HTN) [20] is introduced. It uses method definitions in its planning domain description, which specifies how the complex tasks can be broken down into more manageable tasks [21, 22]. The planning problem can then be specified as a list of tasks to perform. The planner will then solve the problem by applying the breaking down of tasks to every task in the task list. This process continues until the tasks are reduced to their atomic planning operator constituents that correspond to a solution plan. The main disadvantage of this approach lies in the fact that the planning process requires that certain decomposition rules be specified due to its hierarchical nature. This means that it needs to be encoded in advance by an expert.

Table 1: Example activities and their weights.

Name	Precondition	Effect	Weight
generatorON	n/a	$g \coloneqq 1$	2
lightON	$g = 1$	$L \coloneqq 1$	3
fanON	$g = 1$	$f \coloneqq 1$	2

The methods discussed thus far either require exact matchups between inputs, outputs, and variables or assume certain ontologies to handle heterogeneities or require specifications of user intention and procedural templates. Domain and goal modeling through CSP is developed to create a language that allows users to express goals without having to know about the details and interdependencies between services [4, 15, 23]. Its domain representation is of similar concept to the Multivalued Planning Task (MPT) encoding [24]. Besides, another advantage is that it is able to handle variables with large domain efficiently, which is quite prevalent in the field of autonomous home such as temperature value and user location. Although the CSP planner might be slower than some state-of-the-art methods [22, 25, 26], it can support complex goals and can handle variables with large domain efficiently. To be able to support optimization capability, constraints need to be endowed with weights, thus extending CSP to WCSP.

3. The Proposed System

A system to compose complex plans that are optimized is built based on WCSP. This section will first consider the domain description of the problem and how representing the planning problem as CSP and solving it help generate complex plans to achieve goals. The CSP approach described is based on the work of Kaldeli et al. [4]. Subsequently, extension to WCSP through branch and bound method will be explained.

3.1. Planner Module. This section introduces the planning/implementation process devised for performing automated planning and execution under an uncertain situation. Figure 1 shows the flow chart of the planning and plans implementation process, which is central to the planning/implementation block of the ISS framework. With the services that come from devices connected to the ISS, the planner will plan a sequence of actions to be implemented to fulfill goals.

Variables are instances to record knowledge (termed as knowledge variable) or act as a switch pertaining to a corresponding object (termed as effect variable). An example of a knowledge variable is the variable that records room temperature. For effect variable, an example is a variable *TV*, which turns the TV on if it is of state 1.

An activity is a service that is being wrapped with appropriate parameters, effects, and preconditions. Examples are shown in Tables 1 and 2. At planning stage, information of devices associated with the activities is obscured from

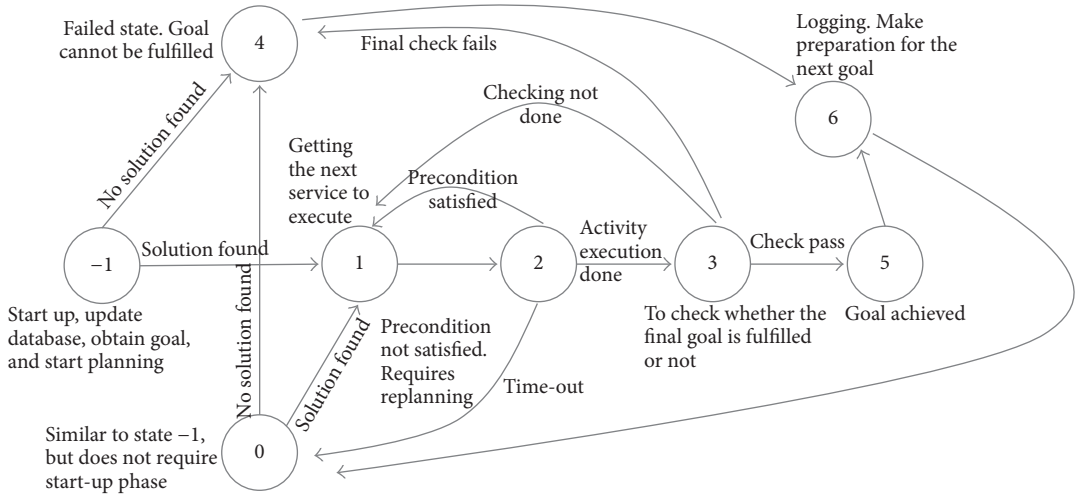

Figure 1: Service composition and execution process flow.

Table 2: Examples of relevant activities (as FOL).

Name	Precondition	Effect	Wt
generator1ON	*true*	$g \coloneqq 1,\ g_{changed} \coloneqq 1$	3
generator1OFF	*true*	$g \coloneqq 0,\ g_{changed} \coloneqq 1$	3
generator2ON	*true*	$g2 \coloneqq 1,\ g2_{changed} \coloneqq 1$	2
generator2OFF	*true*	$g2 \coloneqq 0,\ g2_{changed} \coloneqq 1$	2
lightDim'n'ON	$BrightLight(n) \rightarrow false \wedge \exists_x Generator(x) = 1$	$L(n) \coloneqq 1,\ L_{changed}(n) \coloneqq 1$	$Cost(n)$
lightDim'n'OFF	*true*	$L(n) \coloneqq 0,\ L_{changed}(n) \coloneqq 1$	$Cost(n)$
lightBright'n'ON	$\neg BrightLight(n) \rightarrow false \wedge \exists_x Generator(x) = 1$	$L(n) \coloneqq 1,\ L_{changed}(n) \coloneqq 1$	$Cost(n)$
lightBright'n'OFF	*true*	$L(n) \coloneqq 0,\ L_{changed}(n) \coloneqq 1$	$Cost(n)$
fan'n'ON	$\exists_x Generator(x) = 1$	$f(n) \coloneqq 1,\ f_{changed}(n) \coloneqq 1$	$Cost(n)$
fan'n'OFF	*true*	$f(n) \coloneqq 0,\ f_{changed}(n) \coloneqq 1$	$Cost(n)$
door'n'Open	$Obstruction(n) = 0$	$D(n) \coloneqq 1,\ D_{changed}(n) \coloneqq 1$	2
door'n'Close	$Obstruction(n) = 0$	$D(n) \coloneqq 0,\ D_{changed}(n) \coloneqq 1$	2
window'n'Open	$Swing(n) \wedge Curtain(n, c) \rightarrow Curtain(c) = 1$	$W(n) \coloneqq 1,\ W_{changed}(n) \coloneqq 1$	$Cost(n)$
window'n'Close	*true*	$W(n) \coloneqq 0,\ W_{changed}(n) \coloneqq 1$	$Cost(n)$
curtain'n'Open	*true*	$Cr(n) \coloneqq 1,\ Cr_{changed}(n) \coloneqq 1$	$Cost(n)$
curtain'n'Close	$Curtain(w, c) \rightarrow window(w) = 0$	$Cr(n) \coloneqq 0,\ Cr_{changed}(n) \coloneqq 1$	$Cost(n)$
TVON	*true*	$Tv \coloneqq 1,\ Tv_{changed} \coloneqq 1$	2
TVOFF	*true*	$Tv \coloneqq 0,\ Tv_{changed} \coloneqq 1$	2
SwitchChannel	$UPChannel_{known} = 1 \wedge TV \coloneqq 1$	$TVchan \coloneqq UPchannel,\ TVchan_{changed} \coloneqq 1$	2
GetUPchannel	*true*	$UPChannel \coloneqq UPChannel_{response},\ UPChannel_{known} \coloneqq 1$	2
GetWeather	*true*	$Weather \coloneqq Weather_{response},\ Weather_{Known} \coloneqq 1$	2

the planner. Activity's role is to manipulate the values of the variables, which have correspondence with the physical world associated during variable binding. An activity contains preconditions and effects. Preconditions are conditions that need to be true before the activity can be executed. Effects are the changes that the activity makes when the activity is finished without any glitches or unforeseen circumstances.

The following is the explanation for the process flow shown in Figure 1. The system starts at state −1, where initialization is performed and connection is set up. At this stage, it is required that variables are already bound and activities are ready. The system will restart at stage −1 if there are changes to the number of variables and services due to device connection or disconnection or failure. After initialization, goal is obtained and planning is implemented. If a sequence of activities is found that can fulfill the goal, the system will proceed to stage 1. Otherwise, it will proceed to stage 4, indicating a failure. Stage 1 deals with getting the next activity in sequence to be executed. After the next activity is selected, it will proceed to stage 2, which is orchestration of activities. Before execution occurs, it needs to check whether the precondition is fulfilled, upon which, if not, it will proceed

to stage 0 for replanning. If the precondition is fulfilled, activity is executed by sending command to the relevant devices. After the device finishes executing the activity, the system will proceed to stage 3. Connection loss or device failure is bound to happen, which will delay or stop executing the activity altogether. Given this situation, stage 2 has a time-out threshold, upon which if the threshold is reached, the system will assume that the current plan does not work and proceed to stage 0. Despite being not implemented yet, this event can be logged to record the number of failures a certain device faces. This can help determine the quality of service of the devices, such that it can be used as a weight when selecting services. Stage 3 checks for 2 things: whether there is any activity left for execution (if true, the system will proceed to stage 1 to get the next activity) and if there is none, it will check whether the goal is fulfilled or not. If the final goal is not fulfilled, then it will proceed to stage 4, otherwise, to stage 5, indicating a success. Given stage 4 or stage 5, the system will proceed to stage 6. Stage 6 records a log of past events, obtains a new goal, and proceeds to stage 0. Stage 0 is similar to stage −1 except that it does not need to perform initialization.

3.2. Preliminaries. Domain description for activity planning is based on the work of [15]. We denote ϑ as variable set (list of variables). ϑ contains V variables, which consist of knowledge, effect, and response variables confined by their own domain. Response variables represent information that can only be obtained from objects, information that comes from sources not within ϑ. During planning stage, response variables remain the same throughout all planning sequences and represent an unknown value (thus, initialization constraint is imposed on them). From [15], the response variables can take on whatever value to facilitate constraint satisfaction during planning that employs an optimistic approach (values taken on by response variables are considered true).

A state is a tuple of values to variables at a particular plan implementation sequence with index t that is denoted as $X_t = (X_t^1, X_t^2, \ldots, X_t^V)$, where $X_t^1, X_t^2, \ldots, X_t^V \in \vartheta_t$, confined by their domains denoted by $D^1, D^2, \ldots, D^V$. As there is a finite limit to the number of sequences per plan being planned denoted as K, thus, $0 \leq t < K$. For the current work, domains of the variables remain unchanged over time.

α is the set of activities, where $a = (id(a), precond(a), effect(a)) \in \alpha$. $id(a)$ is the identifier of the activity. There is an additional activity in α that does nothing. It has no preconditions and effects, termed as *Nop*.

$precond(a)$ is the precondition that needs to be met before the activity can be executed, such as the location being known as a precondition to implement the weather forecast web service. Precondition of an activity can be described as follows:

$$\begin{aligned} &precond\,(a) ::= prop \mid precond\,(a) \wedge precond\,(a) \mid \\ &\quad precond\,(a) \vee precond\,(a) \mid \neg precond\,(a) \mid precond\,(a) \\ &\quad \longrightarrow precond\,(a) \\ &\quad prop ::= var \bullet var \mid var \bullet val \mid \\ &\quad (var \odot var) \bullet var \mid (var \odot var) \bullet val \mid Brel, \end{aligned} \tag{1}$$

where $var \in \vartheta$, val is a constant, $\odot \in \{+, -\}$ is a binary operator, $\bullet \in \{=, <, >, \neq, \leq, \geq\}$ is a relational operator, and $Brel$ is a Boolean relation.

$effect(a)$ is the changes that will be induced after the activity is completed. It emulates how the variables will change given the activity is run by its corresponding objects such that its logical formulation can be used to impose constraints on subsequent sequence of the plan for activity planning. It should be emphasized that the actual object manipulates variables during run-time after planning instead of the $effect(a)$ formulation (which is only used for planning). Effect of an activity can be formulated as or a combination of the following: $var_{t+1} = val$, $var_{t+1} = var_t$, and $var_{t+1} = f(v_1, v_2)$, where $v_1, v_2 \in \vartheta_t$ or v_1, v_2 are constants and f is the sum, subtraction, and Boolean operation.

Given the goals, which are represented as propositions, activity planning can be obtained to fulfill the goal by representing the problem as constraint satisfaction problem (CSP) and solve it [4, 15]. A constraint satisfaction problem is a triple CSP $= \langle \chi, D, \zeta \rangle$, where χ is a set of variables, D is the set of domains of the variables in χ, and ζ is a set of constraints over χ. χ consists of every state up to index K, where $\chi = \{X_1, X_2, \ldots, X_K\} \cup \{A_1, A_2, \ldots, A_{K-1}\} \cup R$, R is a set of response variables, and $A_t \in \alpha$ is the chosen goal at sequence index t. A solution to a CSP is an assignment of values to the variables in χ such that the values fall within D and all constraints in ζ are satisfied, which is normally obtained from backtracking methods [27].

ζ consists of constraints imposed by a chosen activity at t from activity preconditions and effects, law of inertia, initial and final variable state, and maintenance of achieved goal constraints. Initial variable state is just a constraint that dictates the values of all variables (obtained from object state module) before any planning. Final state constraint consists of the goal proposition that needs to hold at sequence index K.

Constraints from activity preconditions:

$$(A_t = a) \longrightarrow precond\,(a), \quad \text{where } \forall a \in \alpha. \tag{2}$$

Constraints from activity effects:

$$(A_t = a) \longrightarrow \left[(var_{t+1} = effects_t\,(a)) \wedge Fr\right], \quad \text{where } \forall a \in \alpha, \tag{3}$$

where Fr is the law of inertia, which indicates that for every other variable var (excluding those from R) not affected by $effects_t(a)$, $var_{t+1} = var_t$.

Maintenance of Achieved Goal Constraint. This constraint dictates that whenever a goal is achieved at sequence index $\bar{t} < K$

$$A_t = Nop, \quad \text{where } \bar{t} < t < (K-1). \tag{4}$$

Maintenance of achieved goal constraint is just one of the goals specified in [15, 23], though it is sufficient for the current work.

There are 3 types of variables during planning, which are knowledge, effect, and response variables. Knowledge variable stores a piece of information that can be referred to in the future. Associated with every knowledge variable is a flag that indicates whether the information is updated or not (in this paper, these are variables with subscript *Known*). Effect variables record the state objects are in. Every effect variable has a flag to indicate whether the state is changed or not (the variable is indicated by a subscript *Changed*). Response variables represent information that can only be obtained during run-time. Details on how these variables work can be found in [15, 23].

To increase the speed of the constraint satisfaction solver, only activities that are relevant to the given goals are selected. This work employs the same method used by [28]. Irrelevant activities to the goal are pruned out to reduce the search space. For every activity a_i, a list of other activities that has at least one effect that has the potential to satisfy one of the preconditions of a_i is found. Backward action chain is then computed. This chain of actions can be used to select the activities that are relevant to the imposed goal. R_{act} is denoted as the set of relevant activities. An interesting alternative is to use dependency graph as proposed in this work [29] or to divide all the activities into effects ontology to shortlist relevant activities [30].

CSP approach employs hard constraints, which are fed to a solver to obtain a sequence of A that are the activities that need to be implemented to fulfill the given goals. One example of state-of-the-art solver is Z3 SMT solver, which can efficiently obtain the plan given hard constraints [31]. The plan will be solved by continually increasing K until the constraints are satisfied. Such solvers cannot handle soft constraints to perform optimization and partial fulfillment of goals, or at least not efficiently.

A Weighted Constraint Satisfaction Problem (WCSP) can be described as a tuple $\langle \chi, D, F \rangle$ [32], where $\chi = X_1, X_2, \ldots, X_K \cup R$, D is the set of domains of the variables in χ, and F is a set of weighted constraints. One can think of ζ as F with all the constraints f having infinity as weight values $Weight(f)$. The objective function is the sum of all functions in F:

$$L = \sum_{f \in F} Weight\left(f\right). \tag{5}$$

The goal is to find the instantiation of all variables such that it minimizes the objective function.

Various approaches have been used to solve general WCSP, which includes search, clustering, and variable elimination as explained in this paper [33]. Good heuristics used can also further hasten solving process [34]. Our problem is specific to planning with domain description described previously. Therefore, internal structures can be exploited to build a solver that is specially tailored for plan composition via solving WCSP. Branch and bound with depth first search is used due to polynomial space complexity and the ability to handle constraints of high arity and wide domain, compared to methods like variable elimination [35] (though variable elimination can be combined with search to obtain a much better result [33]). Besides, using a relaxed condition of the former planning problem as shown in Section 3.5, lower bound can be easily obtained to prune the search graph.

3.3. Design of Automated Planner. Given an initial state (or initial variable instantiations), an optimized plan means a plan that can fulfill the maximum number of goals with the least number of activities (or with the least cost given the sum of the costs for all activities), where the sequences of activities need to abide by their individual precondition and effects described in Section 3.2.

Goals are constraints for the final possible variable instantiations. For example, if the goal is $A \vee B$, then final state should be $A_K = 1, B_K = 0$ or $A_K = 0, B_K = 1$ or $A_K = 1, B_K = 1$. Weights associated with every goal are imposed as costs if the goal is not fulfilled in the final state.

Constraints imposed by activities are the preconditions and effects. An activity can only be implemented if its preconditions are met. For example, a TV can only change its channel given the precondition that the TV is on. At the same time, it is considered (at least in planning phase) that effects of activities will definitely change the subsequent state (e.g., the act of changing TV channel will result in a change to the preferred channel). Due to uncertainty, effects might not give the desirable response during actual execution. But this is not an issue given the dynamic planning framework described in Section 3.1, which supports replanning. That said, constraints from preconditions and effects are treated as hard constraints unlike goal imposed constraints.

Activities themselves also come with weights. But unlike weights for constraints, these weights are imposed when an activity is implemented. They act as costs for their respective activities. This is important as there are times when one prefers certain activities to others. For example, during late night, a person might prefer the dimmer light to be switched on instead of the brighter ones. Therefore, the action to switch on the brighter light imposes a higher cost than its dimmer counterparts.

It is assumed that weights are determined depending on situation and time. But this paper does not deal with weight settings. Our intention is that, given a set of constraints and weights, the system will try to generate a sequence of activities as plans that is optimum. In order to achieve that, it needs to minimize the following cost function:

$$A^{opt} = \underset{A_0 \cdot A_1 \cdots A_{K-1}}{\operatorname{argmin}} \left(\left[\sum_{0 \le t < K} C\left(A_t\right) + Pc\left(A_t, S_t\right) \right] + Gc\left(\widehat{A} \circ S_0\right) \right), \tag{6}$$

where $C(A_t)$ is the weight for implementing activity A_t and $Pc(A_t, S_t)$ gives the cost given whether or not precondition for A_t is fulfilled by state S at sequence t. Due to precondition being a hard constraint, Pc will return infinity if precondition is not met.

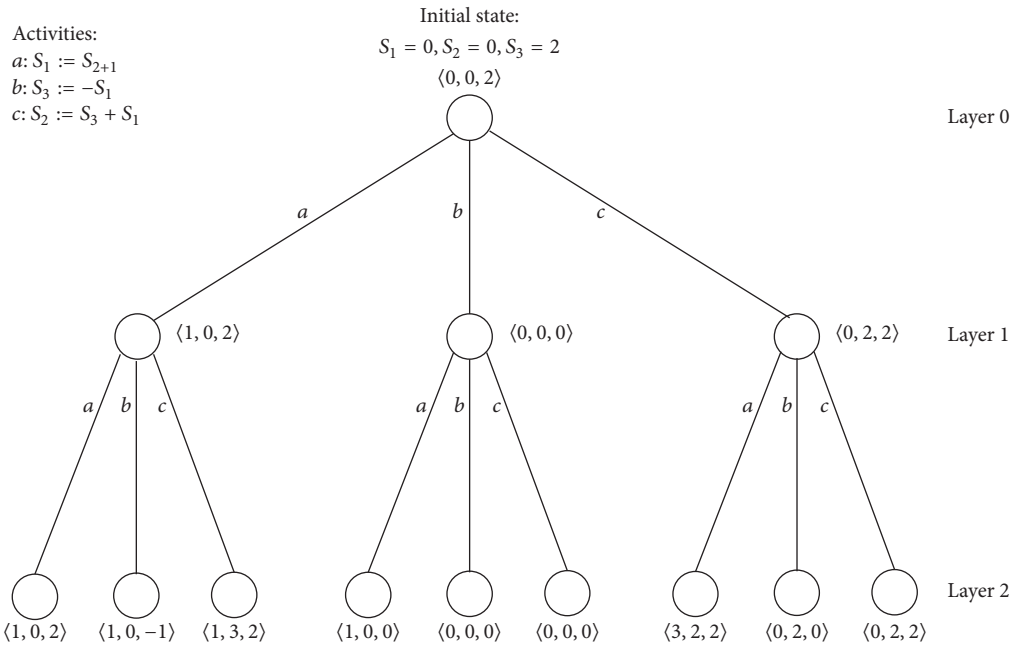

Figure 2: Search tree for planning.

$\widehat{A} \circ S_0 = A_{K-1} \circ A_{K-2} \cdots A_1 \circ A_0 \circ S_0$, which gives the final state S_K when activity A_0 is implemented on state S_0, followed by A_1, and so on until finally A_{K-1}.

$Gc(S) = \sum_{f \in Goal} Weight(f, S)$, which gives the total cost imposed by nonfulfilled goals. $Weight(f, S)$ is different from (5) as it only considers variables at the final state sequence, whereas (5) considers variables of all sequences.

As an example, consider two goals: (1) fan is on $f = 1$ with weight 4 and (2) light is on $L = 1$ with weight 6; and consider an initial state where generator, light, and fan are all off ($g = 0, L = 0, f = 0$, resp., $\langle 0, 0, 0 \rangle$ in short). Example activities are shown in Table 1.

Case 1. Activity sequence is generatorON $\Rightarrow$ lightON $\Rightarrow$ fanON (which means generatorON is executed, followed by lightON and finally fanON). There is no precondition of generatorON; therefore, *Pc* component will return a 0. Effect from the activity will set $g = 1$, thus $\langle 1, 0, 0 \rangle$. Given this new state, precondition for lightON ($g = 1$) is also met, and execution proceeds until fanON. Finally, the state will be $\langle 1, 1, 1 \rangle$. Total cost returned by *Pc* is 0. Total cost of activity is $7 = C(\text{generatorON}) + C(\text{lightON}) + C(\text{fanON})$. Since the 2 goals are fulfilled, *Gc* returns 0 too. Total cost is 7.

Case 2. Activity sequence is generatorON $\Rightarrow$ lightON. Execution of the plan will result in final state $\langle 1, 1, 0 \rangle$. All preconditions are met; thus, *Pc* returns a total of 0. Total cost of activity is 5. Since goal 1 is not satisfied, *Gc* returns 4. Total cost is 9.

Case 3. Activity sequence is lightON. Since the initial state is $\langle 0, 0, 0 \rangle$, precondition is not met. *Pc* returns ∞, rendering subsequent calculation trivial.

Equation (6) tries to find a sequence of activities that is optimal. Due to the use of logical constraints, optimization becomes complicated. Direct search is intractable, aggravated by hard constraints and its weights from *Pr*.

Equation (6) can be relaxed by eliminating *Pc* and, in turn, reformulated as goals. Problem reformulation is explained in Section 3.5. The relaxed optimization problem can then be used to calculate the lower bound for branch and bound via BMA explained in Section 3.6.

The planning representation as well as the heuristics used for optimization in this work assumes nonparallel encoding.

3.4. Branch and Bound Search. Search tree represents the possible paths to reach potential solutions. For our planning problem, every layer of the tree is a particular sequence of activity, which in our case has a total of K layers. Every node is a particular state. Edges are the actions taken, which transform S_t to S_{t+1}. Figure 2 shows an example of the search tree for planning.

Depth first search branch and bound works by choosing a path that gives the best estimated cost, where the best estimated cost of the nodes for every layer except the layer of the leaf nodes is estimated via a heuristic function H, given its previous cost. H estimates the best possible cost to reach the goal state by using the relaxed optimization problem, which combined with previous cost gives the lower bound. A potential solution is reached when the path reaches the leaf nodes. Total cost of every potential solution will be compared with the upper bound, upon which if lower, the upper bound will be updated. Branch and bound reduces the number of search paths by pruning branches of the respective nodes where their lower bound is higher than or equal to the upper bound.

```
Result: A^opt
Upper bound (global variable) UB = ∞;
Current cost Cc = ∞;
Sequence index tc = 0;
State S initialized;
Activity sequence A set to empty;
Store all relevant activities in R_act;
note: Recursion starts
Function BnB(Cc, tc, S, A);
if tc ≥ K − 1 then
    if CostEst(A, S) < UB then
        Update UB;
        A^opt := A;
        [UB, A^opt] := UB_Refine;
    end
else
    for f = all activities in R_act do
        if Preconditions for f are met and LSh_Check(f)
        then
            Act[f] = BMA(tc, S) + C(f) + Cc;
            note: Store the estimated lower bound if activity f is applied.
                  C(f) returns the weight of activity f;
        else
            Act[f] = ∞;
            note: Or else set the lower bound to ∞;
        end
    end
    SList = sort(Act);
    note: Sort Act in ascending order based on their lower bound;
    for Loop through SList do
        note: Looping through the sorted list;
        LB := Lower bound of current SList selection;
        AC := Activity of current SList selection;
        SC := State after the effect of AC;
        Cc2 := Cc + C(SList);
        if LB ≥ UB then
            Break the loop;
        else
            A := append(AC);
            BnB(Cc2, tc + 1, SC, A);
        end
    end
end
```

Algorithm 1: Recursive search for optimized plan.

Algorithm 1 shows a recursive search algorithm for optimized planning. The result of the algorithm is A^{opt}, which is the optimized sequence of plans.

Before the algorithm starts, current cost Cc and upper bound UB are set to ∞. State S is set to initial state values. Set R_{act} is also constructed, which consists of a collection of relevant activities given the goals as described in [28].

The algorithm starts off by checking whether or not the last layer of the tree is reached, upon which if yes (meaning it is a potential solution), it will update UB if the total cost is lower. Further fine tuning can be made by checking for activity redundancy through *UB_Refine*, which checks whether the omission of certain activities will yield better result. Given a sequence of activities, *UB_Refine* will loop through each and calculate whether their omission will produce lower cost. As fast stochastic search is employed in calculating the lower bound, this comes at a price where certain redundant activity will be introduced (given activity cost does not warrant much cost relative to those imposed by goals, which is the situation for our experimentation).

If leaf nodes are not reached, current node is branched out to the subsequent layer, where each branch represents the effect from each activity. Lower bound is estimated for each node of the new layer via BMA that will be explained in Section 3.6. Lower bounds for activities that do not have their preconditions met are assigned ∞. Besides that, further checking is performed by *LSh_Check*. *LSh_Check* checks whether the selection of a certain activity for branching

coincides with pairs listed in the local search lookup table, where the lookup table is explained in Section 3.7. The activities are then sorted according to their lower bounds to determine which activity should be branched out further for evaluation. If *UB* is lower than or equal to an activity's lower bound, that branch and the subsequent branches in the sort list are all pruned out.

Currently, we assume goals to be the desirable final state. Sometimes, extended goals that dictate how a plan is arranged instead of the desirable final state are needed. For example, the generator should not be switched when any electrical appliances are turned on. Algorithm 1 can be extended to support extended goals by treating such goals as preconditions for activities, such that the part *Preconditions for f are met and LSh_Check(f)* in Algorithm 1 can prune out inconsistent activities. The part can also be modified to support weighted extended goals such that unfulfilled extended goals will impose a higher cost on the activity. Extended goal is subject to future work.

3.5. Relaxing Optimization Problem. To obtain a larger feasible region by removing restrictions, (6) is relaxed by converting the $Pc(\bullet)$ component to goals, which gives

$$A^{opt} = \underset{A_0 \cdot A_1 \cdots A_{K-1}}{\operatorname{argmin}} \left(\sum_{0 \le t < K} C\left(A_t\right) + \overline{Gc}\left(\widehat{A} \circ S_0\right) \right). \quad (7)$$

$\overline{Gc}(\bullet)$ varies from $Gc(\bullet)$ in the sense that it considers not only cost from fulfillment of goals, but also extra subgoals introduced by the conversion of $Pc(\bullet)$.

Conversion is done under the concept that whenever an effect of an activity takes place, its precondition has to be met. Therefore, subgoals are constructed in such a way that if a variable holds a certain value in the final state and that it is different from its initial value, it implies that one of the preconditions of activities which subject that particular variable to hold that value has to be true. Currently, only variables of Boolean type are considered for conversion. Other types such as integer and finite domain sorts are subject to future research. Conversion of variables is limited to 4 types of effects pertaining to the variable, which are $V \coloneqq 0$, $V \coloneqq 1$, $V \coloneqq var$, and $V \coloneqq \neg var$, where *var* is a variable. The first two types of conversion are shown in Algorithm 2. The third type is shown in Algorithm 3, whereas the fourth is shown in Algorithm 4. The number of generated subgoals is denoted by *nSG*. Extra subgoals equipped with weights reduce the effective branching factor. Optimizing the relaxed problem thus gives a better lower bound.

3.6. Bacterial Memetic Algorithm Optimization. Bacterial memetic algorithm (BMA) is applied for finding the optimal solution for the relaxed optimization problem discussed in Section 3.5. BMA is a population based stochastic optimization technique which effectively combines global and local search in order to find good quasi-optimal solution for the given problem [36]. In the global search, BMA applies the bacterial operators, the bacterial mutation, and the gene transfer operation. The role of the bacterial mutation is the optimization of the bacteria's chromosome. The gene transfer allows the information's transfer in the population among the different bacteria. As a local search technique, individual reparation is applied for all bacteria.

```
V = val, val is a value where either val = 0 or 1;
V_0 is the initial value of V;
Pstore set to empty;
if V is not a response variable then
    for f = all activities do
        if Found V ≔ val in the f effects then
            Add f precondition in Pstore;
        end
    end
end
Add (V = val ∧ V_0 ≠ V) → (⋁ Pstore) as sub-goal;
note: ⋁ Pstore statement is true if at least one addition of
      f precondition in it is true;
```

Algorithm 2: Conversion for direct value assignment.

```
V = var;
V_0 is the initial value of V;
Pstore set to empty;
if V is not a response variable then
    for f = all activities do
        if Found V ≔ 1 in the f effects then
            Add f precondition in Pstore;
        end
    end
end
Add (V = 1 ∧ var = 1 ∧ V_0 ≠ V) → (⋁ Pstore) as sub-goal;
Add (V = 0 ∧ var = 0 ∧ V_0 ≠ V) → (⋁ Pstore) as sub-goal;
```

Algorithm 3: Conversion for variable assignment.

In case of evolutionary and memetic algorithms, we have to discuss the encoding method and the evaluation of the individuals (bacteria) as well. The encoding of the individual is a sequence with $K-1$ indexes. The evaluation of the bacteria is calculated by

$$Cost = \sum_{0 \le t < K} C\left(A_t\right) + \overline{Gc}\left(\widehat{A} \circ S_0\right). \quad (8)$$

BMA's task is to generate a sequence of activities $\widehat{A}$ (from a pool of activities from R_{act}) such that *Cost* is minimized.

The operation of the BMA starts with the generation of a random initial population containing N_{ind} individuals (see Algorithm 5). Every individual is a particular sequence $\widehat{A}$. Next, until a stopping criterion is fulfilled (which is usually the number of generations, N_{gen}), we apply the bacterial mutation, gene transfer, and local search operators. Bacterial mutation creates N_{clones} number of clones (copies) of an individual, which are then subjected to random changes in their genes (see Algorithm 6). The number of genes that are modified with this mutation is a parameter of the algorithm (l_{bm}). After the bacterial mutation, the gene transfer operation is applied at population level (see Algorithm 7). This means

```
V = var;
V_0 is the initial value of V;
Pstore set to empty;
for f = all activities do
    if f is not a response variable then
        if Found V := ¬var in the f effects then
            Add f precondition in Pstore;
        end
    end
end
Add (V = 1 ∧ var = 0 ∧ V_0 ≠ V) → (⋁ Pstore) as sub-goal;
Add (V = 0 ∧ var = 1 ∧ V_0 ≠ V) → (⋁ Pstore) as sub-goal;
```

Algorithm 4: Conversion for negated variable assignment.

```
Create initial population;
generation ← 0;
while generation ≠ N_gen do
    for i ← 1 to N_ind do
        BacterialMutation(bacterium_i)
    end
    GeneTransfer;
    for i ← 1 to N_ind do
        LocalSearch(bacterium_i)
    end
    generation ← generation + 1;
end
```

Algorithm 5: Bacterial memetic algorithm.

```
Create N_clones + 1 clones of bacterium;
N_segments ← K/l_bm;
for i ← 1 to N_segments do
    Select l_bm yet unmutated random genes;
    for k ← 2 to N_clones + 1 do
        Mutate the selected genes in clone_k
    end
    for k ← 1 to N_clones + 1 do
        Evaluate clone_k using (8)
    end
    Select the best clone;
    Best clone transfers the mutated genes to all clones;
end
bacterium ← best clone
```

Algorithm 6: Bacterial mutation.

```
for i ← 1 to N_inf do
    Ascending order of the population according to (8);
    SourceBacterium ← Random(0, ..., N_ind/2 − 1)
    DestinationBacterium ← Random(N_ind/2, ..., N_ind)
    Select random consecutive l_gt genes;
    Transfer the selected genes from SourceBacterium to
    DestinationBacterium
end
```

Algorithm 7: Gene transfer.

copying genes from better individuals to worse ones. For this reason, the population is split into two halves, according to the cost values. The number of gene transfers in one generation (N_{inf}), as well as the number of genes (l_{gt}) that get transferred with each operation, is determined by the parameters of the algorithm. The last operator in each generation is the local search, which performs reparation of all individuals, using a lookup table, which describes conflicting values in the neighborhood genes of the bacterial chromosome.

Bacterial memetic algorithm has been successfully applied to a wide range of problems. More details about the algorithm can be found in [36, 37].

3.7. Local Search. Combinations of activities that produce trivial effects should be avoided. Examples of such combinations are turning on and then off the lights and turning the light on two consecutive times.

In this work, we only assume combination of two. To obtain a list of such trivial pairs, all activities are paired with other activities, including themselves. Given an initial state, a pair of activities is considered trivial if the state after their execution is the same as the initial state. The list is generated according to the activities available before the search commences.

This list is used by BMA to perform local search and better reparation. The local search list is also used in branch and bound search through *LSh_Check*. It determines whether branching out of a particular activity is redundant given the previous activity, thus reducing search space.

4. Experiments and Discussion

4.1. Setup. Smart home automation is built up from an assortment of devices that are loosely bounded, where they interoperate to provide services. Since they are loosely bound to each other, they do not know anything other than the activities they can provide themselves. Examples of such atomic activities are shown in Table 2. Preconditions and effects are shown as first-order logic (FOL) to save space. In actual implementation, the formulas are all grounded.

Activity *Nop* has weight $wNop = 1$. Every other activity has weight greater than that of *Nop*. This means activity *Nop* is preferred over others after the goal is achieved. Subgoals (not goals) have weight $wSG = 1$. Every goal (not subgoals) has weight $= K \times wNop + nSG \times wSG$. This puts goals as having higher priority than activities and subgoal fulfillment.

Since the purpose of this paper is not on ways to set weights, but to come up with an optimized plan around predefined weights, weights are set for the purpose of experiments. The same case applies to goals. Goals' initial conditions and weights (termed configurations) are specially designed such that optimal plans consist of 4 to 16 steps of activities. These assortments of configurations will be

Table 3: Example activities.

Name	Precondition	Effect	Wt
generator1ON	*true*	$g1 := 1$	3
generator2ON	*true*	$g2 := 1$	2
light'1'ON	$g1 = 1 \vee g2 = 1$	$L1 := 1$	4
light'2'ON	$g1 = 1 \vee g2 = 1$	$L2 := 1$	2
light'3'ON	$g1 = 1 \vee g2 = 1$	$L3 := 1$	3
fan'1'ON	$g1 = 1 \vee g2 = 1$	$f1 := 1$	3
fan'2'ON	$g1 = 1 \vee g2 = 1$	$f2 := 1$	2

randomly chosen given the number of steps of the optimal plan during test.

As an example, assume $K = 10$. Table 3 shows the activities in this example. All formulas are grounded. Boolean term is treated as an integer of 1 and 0.

Goals are as follows:

$$2 = L1 + L2 + L3.$$

$$f1 = 1.$$

$$f2 = 1.$$

$$f2 \neq f1.$$

Initial state is $g1 = 0$, $g2 = 0$, $L1 = 0$, $L2 = 0$, $L3 = 0$, $f1 = 0$, $f2 = 0$.

The optimal solution is

generator2ON $\Rightarrow$ light'3'ON $\Rightarrow$ light'2'ON $\Rightarrow$ fan'2'ON .

The optimal cost is

14(activity cost) + 0(subgoal cost) + 10(goal cost) = 24.

Therefore, the optimal plan is 4 steps with optimal cost 24.

If instead the planner gives the following solution:

generator2ON $\Rightarrow$ light'3'ON $\Rightarrow$ light'2'ON $\Rightarrow$ fan'1'ON,

its cost is now 25. The difference from optimum cost is thus 4.17%.

4.2. Parameter Selection. Parameters for BMA are number of generations N_{gen}, number of bacteria N_{ind}, number of clones N_{clones}, mutation segment length l_{bm}, number of infections N_{inf}, and infection segment length l_{gt}. For speed and simplicity, l_{bm}, N_{inf}, and l_{gt} are all set to 1.

Parameters N_{gen}, N_{ind}, and N_{clones} are selected via a test run with total sequence K of 10, with optimal solution having 4 activity sequences. *UB_Refine*, *LSh_Check*, and local search are not applied in this test. Table 4 shows the planning cost and Table 5 shows the training time for various values of N_{gen}, N_{ind}, and N_{clones}. Only the relevant parameter settings are shown in the tables.

From the tables, N_{gen} of 8 and 10 has a more consistent result with increasing N_{ind}. Besides, general performance is better compared to the lower N_{gen} values. The downside is

Table 4: Planning cost for various N_{gen}, N_{ind}, and N_{clones} values.

N_{gen}	N_{clones}	N_{ind} 3	4	5	6	7
4	2	19.7	20.2	18.6	19.1	18.8
6	2	19.3	18.5	18.7	17.9	18.2
8	2	18.4	18.1	18.2	17	16.2
10	2	17.3	17.2	16.8	17.2	16.3
4	3	18.2	19.5	17.7	18	17.6
6	3	18	17.6	17.6	17.5	17.4
8	3	17.7	17	16.6	17	16.7
10	3	16.4	16.8	16.3	16.2	16.2
4	4	19	18	17.7	17.8	17.4
6	4	17.2	17.1	16.8	17	16.9
8	4	16.6	16.8	16.7	16.7	16.6
10	4	16.7	16.2	16.3	16.4	16.1

Table 5: Planning time (seconds) for various N_{gen}, N_{ind}, and N_{clones} values.

N_{gen}	N_{clones}	N_{ind} 3	4	5	6	7
4	2	5.1	5.3	7.5	8.8	10.3
6	2	7.8	8.5	10.8	14.9	14.5
8	2	8.1	12.9	16.5	19.2	20.3
10	2	13.2	11.8	33.2	19.2	32.9
4	3	6.6	7.4	10.5	11.8	14.1
6	3	8.7	12.2	11.6	23	28.9
8	3	11.9	22.8	19.9	26	32.3
10	3	13.9	24.4	30.2	22.9	25.3
4	4	9	9.1	12.1	13.3	14.6
6	4	10.3	18.7	26.7	20.4	34.2
8	4	19.2	19.9	28.2	25.3	39.4
10	4	20.5	38.5	30.2	34.5	36.5

their time consumption in order to obtain the solution. For the subsequent tests, $N_{gen} = 10$, $N_{ind} = 3$, and $N_{clones} = 3$. From the tables, with $N_{gen} = 10$ and $N_{clones} = 3$, the performance is more consistent and nearer to the optimal solution. N_{ind} will remain at 3 due to time consumption constraint.

4.3. Planning Performance without Time Threshold. Planning test is performed with *UB_Refine*, *LSh_Check*, and BMA local search. As mentioned in Section 3.4, *UB_Refine* will loop through the activity sequence and checks whether any activity's omission will produce better results. *LSh_Check* will check whether branching out of an activity is redundant or not based on the local search lookup explained in Section 3.7. Total sequence K is set to 20. A set of goals is designed such that the number of activities of their optimal plan lies between 4 and 8. Test is performed on 4, 6, and 8 optimal plan sequences (OPS), where the designed goals are randomly selected.

For each OPS, 5 sets of modifications are tested, which are (1) planning without *UB_Refine*, *LSh_Check*, and local search, (2) planning with *UB_Refine* and without *LSh_Check* and local

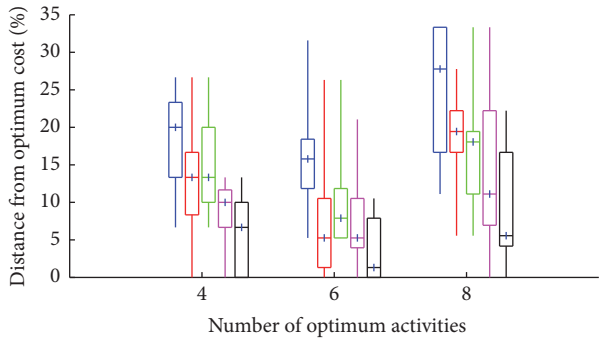

Figure 3: Distance from optimum cost (in percentage) test result for planning without time threshold. Modifications from 1 to 5 (explained in Section 4.3) are indicated by colors blue, red, green, magenta, and black, respectively. The top of the box indicates the 3rd quartile, and the bottom of the box indicates the 1st quartile. The cross indicates the median. Line extension shows the minimum and maximum of the collected test samples. The same indicators apply to subsequent graph.

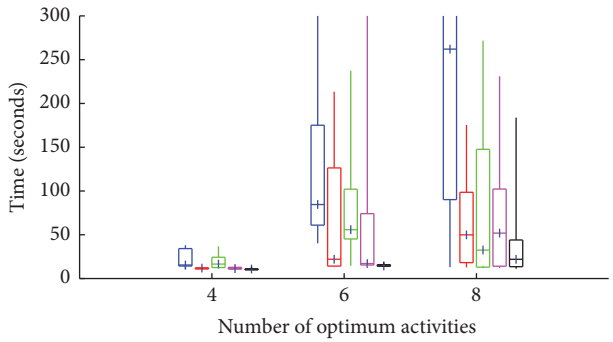

Figure 4: Planning time test result for planning without time threshold.

search, (3) planning with *LSh_Check* and without *UB_Refine* and local search, (4) planning with both *UB_Refine* and *LSh_Check* but without local search, and (5) planning with *UB_Refine*, *LSh_Check*, and BMA local search. Modifications 1 to 4 can be seen as Bacterial Evolutionary Algorithm, while modification 5 is the bacterial memetic algorithm. Each modification is run 20 times. There is no time threshold; therefore, for every test run, the planner will proceed until full search is completed. Figure 3 shows the result on distance from optimum cost, with the tests planning time shown in Figure 4.

It can be observed that inclusion of *UB_Refine* and *LSh_Check* can help improve performance in terms of approaching the optimum cost, although *LSh_Check* has a wider distribution compared to *UB_Refine* if applied individually. With the inclusion of local search, the performance is further enhanced.

In terms of consumption time, as expected, the more the activities required, the slower the planning time. But the inclusion of local search greatly reduces planning time, as this may be due to the fact that BMA has a higher chance of obtaining good combinations given reparation from the local search.

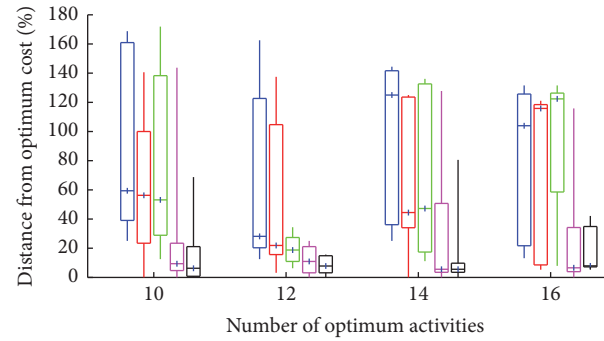

Figure 5: Distance from optimum cost (in percentage) test result for planning with time threshold of 1 minute.

4.4. Planning Performance with Time Threshold. Previous test shows that time consumption is a significant issue, where planning time may reach several minutes. Increasing OPS further becomes impractical.

Alternatively, instead of waiting for the planner to finish, the best possible solution within a certain time limit can be used instead. This is likened to the planner generating the first few moves for execution, before proceeding further to eventually complete the whole process.

To test this idea, a time threshold of 1 minute is used to obtain plans of OPS from 10 to 16. If planning process does not end after 1 minute, the best solution is obtained instead. Experimental procedure is the same as in Section 4.3. Result of distance from optimal cost is shown in Figure 5. Consumption time is not examined as most of the planning process exceeds threshold time.

UB_Refine and *LSh_Check* do not fare so well by themselves. From time consumption result shown in Figure 4, both of them when applied individually require longer searches for OPS of 6 and 8. Cutting them short by introducing a time threshold of 1 minute forces the current best solution to be used, which is suboptimal. Combining them, coupled by the local search, enables the planner to obtain significantly better plans. Spread of distribution of performance is also reduced at significant magnitude, indicating a more consistent performance given local search is employed for gene reparation in BMA.

The performance test for Figure 5 is conducted with the assumption that the optimum plan is to be obtained given only 1 planning being done. The bad performance of the distribution is mainly due to unfulfilled goals (since goals impose the most weights). One planning is fairly restrictive, given that, in real life, planning and execution can be done sequentially. This means planning only needs to come up with partial plan to be executed, before proceeding to the next plan, until all goals are fulfilled, which is obviously more efficient. Such continuous planning is tested with 16 steps of optimal plan. Planning (and execution, which is just the manipulation of state variables) is run until no plans can be devised (which means all the activities in the $K-1$ sequence of activities are *Nop*). Threshold is 1 minute for each planning process. Performance is shown in Table 6. As can be observed, the performance drastically increases. Single planning gives

Table 6: Performance for continuous planning for modifications 1 to 5.

Modification	1	2	3	4	5
Diff. from optimum	30.3%	4.5%	15.4%	2.2%	2.0%

worse result compared to multiple planning because the former produces an incomplete plan. Given multiple implementations of the planner (with execution), it will approach the optimal solution. The bad result of modifications 1 and 3 is due to redundant activities contributing more costs. For other modifications, the main reason why they cannot reach 0% difference from optimum is due to goals that are related to conditions on integer (e.g., $3 \leq L1 + L2 + L3 + L4$, where there are specific activities to set $L1$ to $L4$ 1 or 0 with different weights). As our current subgoals do not take integer into account, the planner does not know whether it is getting nearer or farther away from a solution related to integer. It only considers whether the constraint is fulfilled or not. In this case, it may choose activities with higher costs to fulfill the goal. Future work will involve generating subgoals for constraints involving integers to improve branching factor.

4.5. Comparison with CSP. The use of hard constraints in the CSP planner [4] makes it very inefficient to handle overconstrained situations, where one needs to resort to partial goal fulfillment. Given problems that have contradicting goals, CSP planner will deem it unsolvable. WCSP planner directly assigns weights to the goals, where instead of satisfying constraints it works by maximizing the fulfillment of goals, which subsumes the former. Besides, with too many goals that cannot be solved at one go, WCSP can solve them through continuous planning and execution. It should be emphasized that this advantage holds only if each goal is solvable given the fixed planning horizon, as opposed to one goal that requires long complicated planning solution. For the latter, if the planning horizon is below minimum steps of actions to fulfill the goal, WCSP planner will produce an empty plan due to *UB_Refine* eliminating all actions (which it deemed redundant).

Work in [4] also provides a fixed planning horizon, which introduces redundancy. Redundancy check is only partially realized through the use of constraints that maintain the state upon achieving the goal by imposing the no-action *nop* activity. A workaround is to continually increase the planning horizon until constraint consistency is achieved. For the proposed approach, optimization and upper bound minimization through *UB_Refine* inherently prevent redundancy from occurring given fixed planning horizon.

Besides that, plan optimization is crucial as certain actions are preferable depending on context. For CSP planner, all actions are considered equal, and there is no direct way to encode preferences. As for WCSP, action preferences are encoded as costs, which will then be taken into account together with penalties of unfulfilled goals during optimization.

Despite the advantages, WCSP planner will be substantially slower compared to CSP planner due to it having to find a consistent plan and the fact that the plan needs to be optimized. Therefore, planning horizon should not be too large, while at the same time it should be sufficient to support reasonable number of steps to fulfill goals. A possible enhancement left as future work is to generate more goals that are the consequence of the original set of goals. For example, the goal $x > 5$ entails $x > 4$, $x > 3$, $x > 2$, and so forth. By including these entailments, planning horizon can be cut short, where complex plan can be continually generated.

5. Conclusion

An automated planner for the smart home that can deal with partial goal fulfillment and plan optimization is developed, where planning problem is solved as WCSP that is an extension of planning via CSP [4].

The planning problem itself is a mixture of soft and hard constraints. By exploiting the structure of the planning problem itself, it can be represented as a branch and bound search problem. Lower bound is calculated given a relaxed version of the original problem by exploiting the precondition and effects of the activities of the smart home. The relaxed optimization problem can be readily handled by evolutionary methods, such as the BMA in our case. BMA provides a lot of room for future research especially in the realm of its local search.

Results obtained from planning with and without time threshold demonstrate the applicability of our approach. Besides, the results also show the strength of applying local search for more efficient reparation of potential solutions in the BMA.

Yet, the work is still far from complete. At the moment, conversion of subgoals of the relaxed optimization problem only deals with Boolean data type. More efficient branching factor can be achieved if other data types are taken into account. Apart from that, faster searches and better heuristics should be devised to achieve better planning performance. Given the structure of the problem, search space can also be reduced by applying bucket elimination [38], where functions of low arity are eliminated before passing the constraint graph to search methods.

Reference [39] has argued that smart homes are developed mostly from the viewpoint of technical capabilities. It proposes to shift the smart home paradigm from technological capabilities to that which is centered around humans' need to enhance their living experience. The emphasis is that smart home should function in a way that tries to maximize the fulfillment of goals based on objects connected to it, instead of humans having to decide how the devices are going to serve them. Currently, smart devices that are able to learn and adapt their control are on the rise. Although this work does not involve machine learning or data mining to deliver services, relevancy can be drawn with such smart devices in that it may use their inferred results as knowledge variables. Given the right specification of ontology and functionalities, smart devices' abilities can be utilized with other devices to maximize goals. Such extensions will be made in future work.

Competing Interests

The authors declare that they have no competing interests.

References

[1] S. Chernbumroong, S. Cang, A. Atkins, and H. Yu, "Elderly activities recognition and classification for applications in assisted living," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1662–1674, 2013.

[2] J. Broekens, M. Heerink, and H. Rosendal, "Assistive social robots in elderly care: a review," *Gerontechnology*, vol. 8, no. 2, pp. 94–103, 2009.

[3] M. P. Papazoglou and W.-J. Van Den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007.

[4] E. Kaldeli, E. U. Warriach, A. Lazovik, and M. Aiello, "Coordinating the web of services for a smart home," *ACM Transactions on the Web*, vol. 7, no. 2, article 10, 2013.

[5] S. N. Han, G. M. Lee, and N. Crespi, "Semantic context-aware service composition for building automation system," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 252–261, 2014.

[6] D. Tang, B. Yusuf, J. Botzheim, N. Kubota, and C. S. Chan, "A novel multimodal communication framework using robot partner for aging population," *Expert Systems with Applications*, vol. 42, no. 9, pp. 4540–4555, 2015.

[7] J. E. Gallardo, C. Cotta, and A. J. Fernández, "Solving weighted constraint satisfaction problems with memetic/exact hybrid algorithms," *Journal of Artificial Intelligence Research*, vol. 35, pp. 533–555, 2009.

[8] M. B. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "OM2M: extensible etsi-compliant M2M service platform with self-configuration capability," *Procedia Computer Science*, vol. 32, pp. 1079–1086, 2014.

[9] N. N. W. Tay, A. A. Saputra, J. Botzheim, and N. Kubota, "Service robot planning via solving constraint satisfaction problem," *ROBOMECH Journal*, vol. 3, no. 1, pp. 1–17, 2016.

[10] D. Bonino and F. Corno, "Rule-based intelligence for domotic environments," *Automation in Construction*, vol. 19, no. 2, pp. 183–196, 2010.

[11] D. Bonino and F. Corno, "DogOnt—ontology modeling for intelligent domotic environments," in *The Semantic Webt—ISWC 2008*, A. Sheth, S. Staab, M. Dean et al., Eds., vol. 5318 of *Lecture Notes in Computer Science*, pp. 790–803, Springer, Berlin, Germany, 2008.

[12] M. Ruta, F. Scioscia, E. Di Sciascio, and G. Loseto, "Semantic-based enhancement of ISO/IEC 14543-3 EIB/KNX standard for building automation," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 731–739, 2011.

[13] T. Di Noia, E. Di Sciascio, and F. M. Donini, "Semantic matchmaking as non-monotonic reasoning: a description logic approach," *Journal of Artificial Intelligence Research*, vol. 29, pp. 269–307, 2007.

[14] G. Loseto, F. Scioscia, M. Ruta, and E. Di Sciascio, "Semantic-based smart homes: a multi-agent approach," in *Proceedings of the 13th Workshop on Objects and Agents (WOA '12)*, vol. 892, pp. 49–55, Milano, Italy, September 2012.

[15] E. Kaldeli, E. Warriach, J. Bresser, A. Lazovik, and M. Aiello, "Interoperation, composition and simulation of services at home," in *Service-Oriented Computing*, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds., vol. 6470 of *Lecture Notes in Computer Science*, pp. 167–181, 2010.

[16] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: a decade's overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[17] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web service composition planning with OWLS-Xplan," in *Proceedings of the 2005 AAAI Fall Symposium on Semantic Web and Agents*, pp. 55–62, Arlington, Va, USA, November 2005.

[18] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas, "Semantic awareness in automated web service composition through planning," in *Artificial Intelligence: Theories, Models and Applications*, vol. 6040 of *Lecture Notes in Computer Science*, pp. 123–132, Springer, 2010.

[19] O. Hatzi, D. Vrakas, M. Nikolaidou, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas, "An integrated approach to automated semantic web service composition through planning," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 319–332, 2012.

[20] I. Georgievski and M. Aiello, "An overview of hierarchical task network planning," https://arxiv.org/abs/1403.7426.

[21] E. Sirin, B. Parsia, D. Wu, J. Handler, and D. Nau, "HTN planning for web service composition using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 4, pp. 377–396, 2004.

[22] T. Au, U. Kuter, and D. Nau, "Web service composition with volatile information," in *The Semantic Web ISWC 2005*, Y. Gil, E. Motta, V. Benjamins, and M. Musen, Eds., vol. 3729 of *Lecture Notes in Computer Science*, pp. 52–66, Springer, Berlin, Germany, 2005.

[23] E. Kaldeli, A. Lazovik, and M. Aiello, "Continual planning with sensing for web service composition," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, San Francisco, Calif, USA, August 2011.

[24] M. Helmert, "Concise finite-domain representations for PDDL planning tasks," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 503–535, 2009.

[25] S. Richter and M. Westphal, "The LAMA planner: guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 127–177, 2010.

[26] J. Hoffmann, I. Weber, and F. Kraft, "Sap speaks PDDL," in *Proceedings of the 24th National Conference of the American Association for Artificial Intelligence*, Atlanta, Ga, USA, July 2010.

[27] J. R. Bitner and E. M. Reingold, "Backtrack programming techniques," *Communications of the ACM*, vol. 18, no. 11, pp. 651–656, 1975.

[28] E. Kaldeli, A. Lazovik, and M. Aiello, "Extended goals for composing services," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS '09)*, 2009.

[29] V. Degeler and A. Lazovik, "Dynamic constraint reasoning in smart environments," in *Proceedings of the IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI '13)*, pp. 167–174, November 2013.

[30] F. Corno and F. Razzak, "Real-time monitoring of high-level states in smart environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 7, no. 2, pp. 133–153, 2015.

[31] L. De Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, Berlin, Germany, 2008.

[32] S. Bistarelli, U. Montanari, and F. Rossi, "Semiring-based constraint satisfaction and optimization," *Journal of the ACM*, vol. 44, no. 2, pp. 201–236, 1997.

[33] J. Larrosa, E. Morancho, and D. Niso, "On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study," *Journal of Artificial Intelligence Research*, vol. 23, pp. 421–440, 2005.

[34] J. H. Moreno-Scott, J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "Experimental matching of instances to heuristics for constraint satisfaction problems," *Computational Intelligence and Neuroscience*, vol. 2016, Article ID 7349070, 15 pages, 2016.

[35] R. Dechter, "Bucket elimination: a unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, 1999.

[36] J. Botzheim, C. Cabrita, L. T. Kóczy, and A. E. Ruano, "Fuzzy rule extraction by bacterial memetic algorithms," *International Journal of Intelligent Systems*, vol. 24, no. 3, pp. 312–339, 2009.

[37] J. Botzheim, Y. Toda, and N. Kubota, "Bacterial memetic algorithm for offline path planning of mobile robots," *Memetic Computing*, vol. 4, no. 1, pp. 73–86, 2012.

[38] J. Larrosa and R. Dechter, "Boosting search with variable elimination in constraint optimization and constraint satisfaction problems," *Constraints*, vol. 8, no. 3, pp. 303–326, 2003.

[39] G. Leitner, "The future home is wise, not smart," in *The Future Home is Wise, Not Smart*, R. Harper, Ed., Computer Supported Cooperative Work, 2015.