Applying automated verification to optimal sizing of stand-alone solar photovoltaic systems

Alessandro Trindade
Electricity Department
Federal University of Amazonas
Manaus, Brazil
alessandrotrindade@ufam.edu.br

Lucas Cordeiro
School of Computer Science
The University of Manchester
Manchester, England
lucas.cordeiro@@manchester.ac.uk

***Abstract*—Currently around 1 billion of people do not have electrical energy. Energy poverty alleviation has become an important political issue. Several initiatives and policies have been proposed to deal with poor access to sources of energy in many developing countries. Particular attention is given to use renewables and stand-alone solar photovoltaic (PV) systems. Validation tools are available to evaluate PV designs, but they are mainly based on simulations that do not cover all aspects of the design space. The use of automated verification in PV systems is a recent subject with a few initiatives. Here we create an algorithm and use four state-of-art automated verification techniques to optimize the sizing of a solar photovoltaic system, where commercial equipment data is part of the specification of the code. Furthermore, the technical solution is conjugated with the financial analysis of the sizing obtained. The proposal is unprecedented and interesting to streamline the design and analysis of photovoltaic solar systems projects.Experimental results show promising results, even with state explosion and time to computing issues not solved yet.**

***Index Terms*—automated verification, model checking, optimization, electrical systems, solar photovoltaic systems**

I. Introduction

Lack of access to clean and affordable energy is considered a core dimension of poverty [**?**]. Progress has been made worldwide; in particular, the number of people without electricity access fell below 1 billion threshold for the first time in 2017 [**?**]. In order to provide universal electricity for all, decentralized systems led by solar photovoltaic (PV) in off-grid and mini-grid systems will be the lowest-cost solution for three-quarters of the additional connections needed; and grid extension will be the standard especially in urban areas [**?**].

In order to simulate or evaluate a PV system, there are various specialized tools available in the market, e.g. RETScreen, HOMER, PVWatts, SAM and Hybrid2 [**?**], [**?**], [**?**], [**?**], [**?**]; and even general purpose simulation tools, e.g. PSpice, Saber or MATLAB/Simulink package [**?**], [**?**]. However, those tools are based on simulation models, which are employed before the system design is concluded; it has the drawback of an incomplete coverage since verification of all possible combinations and potential failures of a system is unfeasible [**?**].

Formal methods based on *model checking* can offer a great potential to obtain a more effective design process of PV systems [**?**]. In this study, we review verification tools and we do a link to PV systems, what was not common in past research.

To select an optimum combination to meet sizing constraints, it is necessary to evaluate *power reliability* and *system cost* analysis for the recommended system. An ideal combination for any PV system is made by the best compromise between two considered objectives, which is power reliability and system cost [**?**].

Regarding power reliability, the usual is to use loss of load probability (LOLP) or loss of power supply probability (LPSP). Based on the fact that at this paper we are not considering site characteristics and neither the load changes over the time, the reliability analysis will be granted only by the critical period method of PV sizing.

In the other hand, considering the system cost analysis, it is usual to find related research done with Net Present Cost (NPC), Levelized Cost of Energy (LCOE) or Life Cycle Cost (LCC). Here, one more time, based on the fact that the local of deployment is not specified, this research uses an adapted LCC analysis, where only the deployment cost is considered at the model (without the maintenance or operational costs).

The main contribution of this approach is to allow that a list of PV components (PV panels, charge controllers, inverters and batteries) can be inputted to the system, and the algorithm using automated verification, and written in language C, can find the best technical and financial result.

II. Automated Verification Using Model Checking

Although simulation and testing explore possible behaviors and scenarios of a given system, they leave open the question of whether unexplored trajectories may contain a flaw. Formal verification conducts an exhaustive exploration of all possible behaviors; when a design is said to be "correct" by a formal verification method, it implies that all behaviors have been explored, and questions regarding adequate coverage or missed behavior becomes irrelevant [**?**]. Formal verification is a systematic approach that applies mathematical reasoning to obtain guarantees about the correctness of a system; one successful method in this domain is model checking [**?**]. Here we evaluate four state-of-the-art model checkers to formally optimize the sizing of PV designs w.r.t. user requirements.

A. CBMC

The C Bounded Model Checker (CBMC) falsifies assertions in C programs or proves that they are safe if a completeness threshold is given [**?**]. CBMC implements a bit-precise translation of a C program, annotated with assertions and with loops unrolled up to a given depth, into a logical formula. If the formula is satisfiable, then a failing execution that leads to a violated assertion exists [**?**]. CBMC's verification flow can be summarized in three stages: (i) Front-end: scans, parses and type-checks C code; it converts control flow elements, such as *if* or *switch* statements, loops and jumps, into equivalent guarded *goto* statements, thus aiming to reduce verification effort; (ii) Middle-end: performs symbolic execution by eagerly unwinding loops up to a fixed bound, which can be specified by the user on a per-loop basis or globally, for all loops and finally; (iv) Back-end: supports SAT and SMT solvers to discharge verification conditions.

B. ESBMC

ESBMC (or Efficient SMT-based Bounded Model Checker) is a bounded and unbounded mdoel checker for C programs [**?**], which supports the verification of LTL properties with bounded traces [**?**]. ESBMC's verification flow can be summarized in three stages: (i) a front-end that can read and compile C code, where the system formal specification is first handled; (ii) preprocessing steps to deal with code representation, control flow and unwinding of loops, and model simplification, thereby aiming to reduce verification effort; and finally (iii) the SMT solving stage, where all constraints and properties of the system are encoded into SMT and checked for satisfiability.

C. Ultimate Automizer

Ultimate Automizer is an automatic software verification tool for C programs that is able to check safety and liveness properties. The tool implements an automata-based instance of the Counterexample-Guides Abstraction Refinement (CEGAR) scheme. CEGAR is a technique that iteratively refines an abstract model using counterexamples and use three concepts: precision, feasibility check and refinement [**?**]. In each iteration, Automizer picks a trace (sequence of statements) that leads from the initial location to the error location and check whether the trace is feasible or infeasible. If the trace is feasible, an error is reported to the user; otherwise it is computed a sequence of predicates along the trace as a proof of the trace's infeasibility. The predicates are a sequence of interpolants since each predicate "interpolates" between a set of reachable states and a set of states from which it cannot reach the error. In the refinement step of the CEGAR loop, Automizer tries to find all traces whose infeasibility can be shown with the given predicates and subtract these traces from the set of (potentially spurious) error traces that have not yet been analyzed. It is used automata to represent sets of traces. The difference to a classical CEGAR-based predicate abstraction is that Automizer has to do any logical reasoning (e.g., SMT solver calls) that involves predicates of different CEGAR iterations [**?**].

D. CPAchecker

Automatic program verification requires a choice between precision and efficiency. The more precise a method, the fewer false positives it will produce, but also the more expensive it is, and thus applicable to fewer programs. Historically, this trade-off was reflected in two major approaches to static verification: program analysis and model checking. In order to experiment with the trade-off, and in order to be able to set the dial between the two extreme points, Configurable Program Analysis (CPA) provides a conceptual basis for expressing different verification approaches in the same formal setting. The CPA formalism provides an interface for the definition of program analyses. Consequently, CPAchecker provides an implementation framework that allows the seamless integration of program analyses that are expressed in the CPA framework. The comparison among different approaches in the same experimental setting is intended to be easy and the experimental results are expected to be more meaningful [**?**]. Related to the architecture, the central data structure is a set of control-flow automata (CFA), which consists of control-flow locations and control-flow edges. The CPA framework provides interfaces to SMT solvers and interpolation procedures [**?**]. Currently, CPAchecker uses MathSAT as SMT solver; and CSIsat and MathSAT as interpolation procedures [**?**].

III. Sizing Stand-alone Solar PV Systems

The modeling of the PV system is based on modular blocks, as illustrated in Fig.**??**. The PV generator, which can be a panel or an array, is a semiconductor device that can convert solar energy into DC electricity. In Fig.**??**, there are two variables that depend on the site where the system is deployed and the weather (i.e., solar irradiance G and temperature T). For night hours or rainy days, we need to hold batteries where power can be stored and used. The use of battery as a storage form implies the presence of a charge controller [**?**]. The PV arrays produce DC and therefore when the PV system contains an AC load, a DC/AC conversion is required. That converter is called of inverter; and the AC load dictates the behavior of AC electrical load from the house that will be fed by the system.

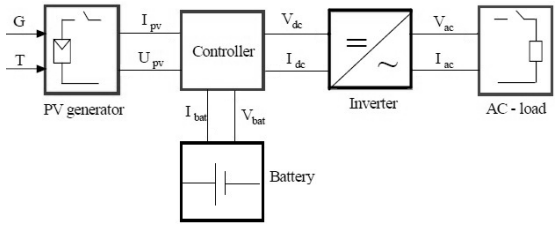

Fig. 1. Block diagram for a typical stand-alone PV system [**?**].

The sizing check will be performed by the proposed tool, and this stage ensures that the system meets the standard

project steps related to critical period solar energy method [**?**] and adopting MPPT (maximum power point tracking) charge controller, that is the most common nowadays. Firstly, it is needed to correct the energy consumption estimated to the load ($E_{consumption}$), which is carried out by (**??**), where the efficiency of batteries (η_b), controller (η_c), and the inverter (η_i) are considered [**?**].

$$E_{corrected}=\frac{E_{consumption}}{\eta_b \eta_c \eta_i}. \quad (1)$$

Following, it is necessary to estimate the energy that can be produced for each panel, called E_p, in Wh, as defined by (**??**).

$$E_p=Solar\_Irradiance\times Panel\_Area\times\eta_p\times 1000, \quad (2)$$

where the solar irradiance is expressed in terms of kWh/m^2 and depends on the site where the PV systems will be deployed, the PV panel area is given in m^2 and corresponds to the size of one PV panel, and η_p represents the PV panel efficiency.

The total minimum number of needed solar panels (N_{TPmin}) is computed by (**??**).

$$N_{TPmin}=\frac{E_{corrected}}{E_p}. \quad (3)$$

Particularly, the total number of panels in series (N_{PSmin}) and parallel (N_{PPmin}) are given by (**??**) and (**??**), respectively. Where $V_{mppt,max}$ is the maximum operation voltage and $V_{mppt,min}$ is the minimum operation voltage of the charge controller; $V_{maxPower,TempMax}$ and $V_{maxPower,TempMin}$ are the maximum power voltage from the PV module considering the maximum and minimum operational temperature, respectively; P_{total} is the total power demanded from the PV system and $P_{max,ref}$ is the power supplied from one PV panel in $Watts$.

$$\frac{V_{mppt,min}}{V_{maxPower,TempMax}}\leq N_{PSmin}\leq\frac{V_{mppt,max}}{V_{maxPower,TempMin}}. \quad (4)$$

$$N_{PPmin}=\frac{P_{total}}{Number\ Panels\ Series\times P_{max,ref}}. \quad (5)$$

Related to the batteries, firstly must be defined the total capacity of the battery bank, as described by (**??**) as

$$C_{bank}=\frac{E_{corrected}\times autonomy}{V_{system}\times DOD}. \quad (6)$$

where the variable $autonomy$ is a design definition; V_{system} is the DC voltage of the bus, and DOD is the battery deep of discharge (considered of maximum of 25% at this paper). Secondly, the total (minimum) number of batteries is computed, as described by (**??**).

$$N_B total=N_{BS}min\times N_{BP}min=\frac{V_{system}}{V_{bat}}\times\frac{C_{bank}}{1\ Battery\ Capacity}. \quad (7)$$

Related to the charge controller, initially it must meet the voltage requirement of the PV system, as described by (**??**) to the charge controller voltage:

$$V_c=V_{system}. \quad (8)$$

Following, the short circuit reference information from the manufacturer's solar panel must be corrected to the cell temperature, as described by (**??**):

$$I_{sc,amb}=\frac{G}{G_{ref}}[I_{sc,ref}+\mu_I\times(T-25)]. \quad (9)$$

The controller must meet the maximum current from the PV array given by (**??**) and (**??**).

$$I_{c,min}=I_{sc,amb}\times N_{PP}. \quad (10)$$

$$I_c\geq I_{c,min}. \quad (11)$$

The sizing project check of the inverter is carried out by means of three equations. Equation (**??**) ensures that the input voltage of the controller meets the system voltage. Equation (**??**) ensures that the output voltage of the controller meets the AC voltage of the load. Finally, (**??**) ensures that the controller can support the total demand of the load ($Demand$) and the surge power (P_{surge}). Where $V_{in}DC$ is the nominal input voltage and $V_{out}AC$ is the nominal output voltage of the inverter; $MAX_{AC,ref}$ is the peak power that the inverter can support.

$$V_{in}DC=V_{system}. \quad (12)$$

$$V_{out}AC=V_{AC}. \quad (13)$$

$$\left[(Demand\leq P_{AC,ref})\ and\ (P_{surge}\leq MAX_{AC,ref})\right]. \quad (14)$$

IV. Method

Algorithm **??** describes the pseudo-code used to perform the optimization of stand-alone PV systems using automated verification.

Algorithm 1 Optimization algorithm

```
initialize variables
declare list of PV panels data and cost
declare list of controllers data and cost
declare list of batteries data and cost
declare list of inverters data and cost
declare the maximum cost possible MaxCost
declare power demand, power peak, energy consumption
declare battery autonomy, deep of discharge, AC voltage
for HintCost = 0 to MaxCost do
    declare non − deterministic variable to select Panel from list
    declare non−deterministic variable to select Controller from list
    declare non − deterministic variable to select Battery from list
    declare non−deterministic variable to select Inverter from list
    calculate E_corrected, E_p
    calculate N_TPmin, N_PSmin, N_PPmin
    calculate C_bank
    calculate N_BSmin, N_BPmin, N_Btotal
    requeriment enforced by assume(V_c)
    calculate I_sc,amb
    calculate I_c,min
    requeriment enforced by assume(I_c)
    requeriment enforced by assume(V_in DC)
    requeriment enforced by assume(V_out AC)
    requeriment enforced by assume(Demand)
    requeriment enforced by assume(P_surge)
    non − det variables hold feasible equipment and cost
    F_obj ← N_TP * Panel_Cost + N_TB * Battery_Cost +
    Controller_Cost + Inverter_Cost
    Violation check with assert(F_obj ≥ HintCost)
end for
return ( )
```

The algorithm starts with the input of manufacturers data and prices of PV panels, batteries, charge controllers and inverters (Lines 2 to 5). Following is necessary to define user requirements (house requirements and design definitions), from Line 6 to 8.

The loop-FOR started at line 9 controls the try to get the lowest cost to the PV solution. It starts with cost of $zero$ and stops only when the algorithm finds a feasible solution which cost breaks the $assert$ of Line 28. When this happens, then the algorithm found the optimum solution and we say that the automatic verification reached a *SAT* (satisfiability) condition. The algorithm uses non-deterministic variables to pick one specific data from a list of PV panels, controllers, batteries and inverters (Lines 10 to 13). That procedure ensures that the automated verification will try all the possibilities of item from each equipment, and combine them in order to mount a feasible PV solution that meet the user requirements.

Next, using the equations described at Section **??** are calculated the sizing necessary variables (Lines 14 to 20). The directive *assume* (from Line 21 to 25) ensure the compatibility of the choose item from the list of equipment: the automatic verification uses only the item (among all the possible ones) that satisfies the statements from Line 14 to 20, and more the Line 18. Therefore, the algorithm reaches the Line 27 with one feasible solution, and the cost of that solution is calculated in F_{obj}.

If the algorithm do not find a feasible solution among the item of equipment that were inputted to the code, then the result is an *UNSAT*, i.e., the program finish and do not find an solution. It indicates that was not possible to combine the itens of each equipment and create a feasible solution.

V. Results and Discussion

We have performed two case studies to evaluate the proposed approach of optimization as described in Table **??**. Furthermore, four start-of-art verification tools, as described in Section **??**: CBMC[1], ESBMC[2], UAutomizer[3], and CPAchecker[4], were used to compare the approach effectiveness and efficiency. The case studies are related to a stand-alone PV system with the specifications shown in Table **??**.

All experiments were conducted on an otherwise idle Intel Xeon CPU E5-4617 (8-cores) with 2.90 GHz and 64 GB of RAM, running Ubuntu 16.04 LTS 64-bits. The experiments were performed with a predefined timeout of 240 minutes.

[1]Command-line: $ cbmc --unwind 100 filename.c --trace

[2]Command-line: $ esbmc filename.c --no-bounds-check --no-pointer-check --unwind 100 --boolector

[3]Command-line: $ ./Ultimate -tc config/AutomizerReach.xml -s config/svcomp-Reach-32bit-Automizer_Default.epf -i filename.c --traceabstraction.limit.analysis.time 900 --traceabstraction.stop.after.first.violation.was.found false --cacsl2boogietranslator.overapproximate.operations.on.floating.types false -- cacsl2boogietranslator.assume.nondeterminstic.values.are.in.range false --rcfgbuilder.add.additional.assume.for.each.assert true --rcfgbuilder.simplify.code.blocks true --rcfgbuilder.size.of.a.code.block LoopFreeBlock

[4]Command-line: $ scripts/cpa.sh -heap 64000m -config config/bmc-incremental.properties -spec config/specification/sv-comp-reachability.spc filename.c

TABLE I
Case studies and Results: optimization of PV systems.

	Case 1	Case 2
Tool	Demand = 501W Peak = 501W Energy=3900Wh/day Battery autonomy=48h	Demand = 915W Peak = 980W Energy=4880Wh/day Battery autonomy=6h
CBMC 5.11 (MiniSat 2.2.1)	Out of Memory	Out of Memory
ESBMC 6.0.0 (Boolector 3.0.1)	Timeout	Timeout
UAutomizer 0.1.24 (Z3 4.8.3)	UNKNOWN	UNKNOWN
CPAchecker 1.8 (MathSAT 5.5.3)	SAT (65 minutes)	x4