Towards Pareto-Optimal Parameter Synthesis for Monotonic Cost Functions

B. Bittner, M. Bozzano, A. Cimatti, M. Gario, A. Griggio
Fondazione Bruno Kessler, Trento, Italy
Email: surname@fbk.eu

***Abstract*—Designers are often required to explore alternative solutions, trading off along different dimensions (e.g., power consumption, weight, cost, reliability, response time). Such exploration can be encoded as a problem of parameter synthesis, i.e., finding a parameter valuation (representing a design solution) such that the corresponding system satisfies a desired property. In this paper, we tackle the problem of parameter synthesis with multi-dimensional cost functions by finding solutions that are in the Pareto front: in the space of best trade-offs possible. We propose several algorithms, based on IC3, that interleave in various ways the search for parameter valuations that satisfy the property, and the optimization with respect to costs. The most effective one relies on the reuse of inductive invariants and on the extraction of unsatisfiable cores to accelerate convergence. Our experimental evaluation shows the feasibility of the approach on practical benchmarks from diagnosability synthesis and product-line engineering, and demonstrates the importance of a tight integration between model checking and cost optimization.**

I. Introduction

Many application domains can be described in terms of *parameterized* systems, where parameters are variables whose value is invariant over time, but is only partially constrained. Choosing an appropriate value of the parameters is a widely spread engineering problem, a form of design space exploration where the parameters can represent different design or deployment decisions. Examples of domains that require the analysis of various solutions include function allocation [1], [2], automated configuration of communication media (e.g., time-triggered ethernet protocols [3], flexray [4], [5]), product lines [6], dynamic memory allocation [7], schedulability analysis [8], and sensor placement [9], [10]. In fact, finding an appropriate valuation is rarely sufficient. Often designers are interested in finding *the most appropriate* valuation with respect to weight, latency, memory footprint, flexibility, reliability. Even more interestingly, there are cases where several of the above dimensions must be taken into account at the same time, and it may be necessary to trade off according to multiple cost functions.

In this paper we consider the problem of parameter synthesis when multiple cost functions cannot be easily combined into one. For example, it is possible that a configuration that is best in terms of reliability (e.g., due to redundancy) may not be optimal in terms of weight, cost, or response times. The solution is to build the so-called Pareto front [11], that is the set of parameter valuations that cannot be improved along one dimension without increasing the cost along the others[1].

We present several algorithms for the construction of the Pareto front on the space of parameter valuations that satisfy a parameterized model checking problem. We remark that we focus on *universal* parameter valuations, that guarantee the satisfaction of a property *for all* associated execution traces: this means that it is not sufficient to analyze a single trace (e.g., constructed by a bounded model checker) to have a guarantee that the parameter valuation is valid.

We tackle the problem under the assumption of monotonicity, that naturally occurs in several domains of practical interest, such as sensor placement [10], product lines engineering [6] and fault-tree analysis [12]. In particular, we require that (i) the space of parameters is upward-closed with respect to property satisfaction, and (ii) the cost functions are monotonic. We propose several algorithms of increasing strength. The first idea is to proceed by *valuations-first*, i.e. to identify the set of all valuations that satisfy the property, and then, within this set, represented as a formula in the parameter variables, to identify the ones on the Pareto front. The upfront computation of the set of valid parameter valuations, corresponding to the first phase, can be tackled in various ways. One way is to carry out a symbolic reachability in the parameterized transition system, e.g., by means of a BDD-based model checker [13]. The scalability of BDD-based techniques is however rather limited. An alternative approach is to solve the existential parameter valuation problem for the negation of the property and then complement. This can be easily encoded on top of a SAT-based engine, where the parameters are free. Once the set of valuations is found, we can independently optimize the complement set [14]. Unfortunately, this approach does not allow us to exploit the cost function for pruning.

The second approach, referred to as *one-cost slicing*, prioritizes the search according to one cost function. The first step is to identify a target value, and to collect all the valid parameter valuations. Then, the valuation with the best value along the other cost functions is selected and further optimized, so that one point in the Pareto front is found. The process is iterated until the limit target value is reached. The monotonicity assumption guarantees that the search can be suitably initialized. Compared to the previous one, this

[1]More formally, the Pareto front of a set of parameter valuations is the subset composed by those valuations associated with cost vectors that are not strictly dominated by any other solution. One valuation γ strictly dominates (or "is preferred to") a valuation γ' if each value of γ is not strictly greater than the corresponding value of γ', and at least one value is strictly less. That is, $\gamma_i \leq {\gamma'}_i$ for each i, and $\gamma_i < {\gamma'}_i$ for some i. This is written as $\gamma \prec \gamma'$ to mean that γ strictly dominates γ'. Then the Pareto frontier is the set of points from Γ that are not strictly dominated by any other point in Γ.

ISBN: 978-0-9835678-4-4. Copyright owned jointly by the authors and FMCAD Inc.

approach needs not wait until all the valid parameter valuations are found; however, it still relies on the computation of the valuations for the selected slice.

The third approach, referred to as *costs-first*, is conceptually rather simple. It is based on a sampling of the space of cost values, and for each of them, on solving the associated (non-parameterized) ground model checking problem. This approach, apparently quite naïve, turns out to be extremely efficient once appropriately cast in the setting of IC3 [15]. Intuitively, rather than solving the ground problem, we solve the parameterized problem under assumptions. When IC3 successfully terminates, as a byproduct it produces a parameterized inductive invariant, possibly containing the assumptions, that is sufficiently strong to prove the property. From the validity proof, we extract an unsatisfiable core that allows us to reduce the candidate set of parameters. This step has a substantial effect in speeding up the search, by accelerating it towards (potentially) less expensive parameter configurations. Another advantage is in the fact that the approach works directly in the space of good parameters, and is thus providing an "any-time" algorithm, that can return a subset of the Pareto front if run only within a given resource bound.

We experimentally evaluated the approach, working on benchmarks from various sources [10], [16]. The results show a significant speed up with respect to methods based on enumeration of violations, both in terms of one cost function, and in the case of multiple cost functions. Incidentally, we also report substantial scalability improvements in significant practical cases, compared to a BDD-based approach previously used for single-cost optimization [10].

Structure of the Paper: This paper is structured as follows. In Section II we review some related work. In Section III we define the spectrum of problems. In Section IV we define the various solutions, and in Section V we discuss the impact of IC3 specific techniques. In Section VI we present two motivating domains. In Section VII we evaluate experimentally the three approaches. In Section VIII we draw some conclusions, and outline some directions for future work.

II. Related work

There are many works dealing with parameter synthesis and parameter optimization. The literature can be classified along various dimensions: discrete parameters versus continuous parameters; combinational (e.g., SMT) problems versus sequential (e.g., reachability) problems; number and quality of parameter valuations found (one vs all valuations vs the optimal ones).

a) MaxBMC: The work closest to ours is [17], where the Pareto front is synthesized in the case of circuit initialization. An initialization sequence is intended to take the circuit, starting from any configuration that it could assume at power-up, to a configuration where all flops are initialized. The work in [17] analyzes the trade-off between two dimensions, i.e., the length of the initialization sequence, and the number of flops initialized after the execution.

There is a key difference with our work: in [17] it is sufficient to find a suitable trace to have a valid parameter valuation (i.e., that satisfies the property), even though it may not be optimal with respect to costs. In this sense, the parameters are *existential* with respect to the traces of the transition system being analyzed. Thus, the framework of Bounded Model Checking can be directly used to find candidate valuations. In our work, however, the parameters are *universal* with respect to the traces: in order to prove the validity of a candidate parameters valuation, a full model checking problem needs to be solved. As a consequence, it is not possible to leverage the "trace finding" mechanism of BMC to generate valid candidate valuations. Other differences are in the fact that in [17] the problem does not fall within the hypothesis of monotonicity, and that our algorithms rely on an extension of with IC3, whereas [17] is based on a complete version of Bounded Model Checking.

b) Combinational Pareto front: Other works on the construction of the Pareto front in a formal setting are [18], [7]. The key difference between these works and the one presented here is in the fact that the problem is combinational (e.g., satisfiability) in nature, while we deal with a sequential problem, i.e., invariant checking for a parameterized transition system. The work in [18] tackles the computation of the Pareto front with respect to cost functions imposed on a combinatorial SMT problem. The work in [7] tackles the problem of Pareto-optimal solutions for the problem of dynamic memory allocation.

c) Real-values parameter synthesis: The work in [8] deals with the synthesis of parameters over continuous ranges, to identify the space of valuations that result in a schedulable set of tasks. The method is based on complement, i.e., the set of bad valuations is computed first, and then complemented. The work in [14] relies on IC3 to solve the same problem more generally and more efficiently. Other works [19], [20], [21], [22], [23] synthesize parameters for real-time and hybrid systems. The key difference with respect to the problem tackled here is that no cost functions are considered, i.e., the space of *all* valuations is considered.

d) Synthesis of Fault trees: The work in [13] proposes several approaches to automatically generate Fault Trees for parameterized transition systems. This can be seen as a sequential problem of discrete parameter synthesis under the hypothesis of monotonicity. The key difference is that in [13] costs are not taken into account - *all* parameter valuations, each representing fault combinations in which a property is falsified, are found. We also remark that the parameters are existential, i.e., a valuation is deemed valid by the existence of a trace.

e) Synthesis of Observability Requirements: Identifying sufficient sets of sensor that guarantee the diagnosability of properties of interest is tackled in [9] and in [10]. Optimizations are found with respect to a single cost function, so there is no notion of Pareto front. The work in [9] proposes an explicit-state exploration of the space of costs, to synthesize a minimal configuration that is a global minimum. Domain specific techniques for the analysis of the sensor placement problem are also discussed.

III. Problem Description

We consider transition systems of the form $S = (X, I, T)$, where X is the set of state variables, $I(X)$ is the initial condition, and $T(X, X')$ is the transition relation.

We generalize transition systems to parametric transition system $S = (U, X, I, T)$, where U is the set of parameters, X is the set of state variables, $I(U, X)$ is the initial condition, and $T(U, X, X')$ is the transition relation. Intuitively, a parameter can be seen as a variable that does not change over time.

We assume that the parameters are Boolean, and valuations are in $\Gamma = \mathbb{B}^{|U|}$. The order relation $<$ over $\mathbb{B}$ induces a partial order $\preceq$ over the parameter valuations Γ.

A valuation to the parameters, γ, provides us with a transition system $S_\gamma = (X, I(\gamma, X), T(\gamma, X, X'))$, in which each parameter is replaced with the value assigned in γ.

We assume the usual symbolic representation: a state is represented as an assignment to the X variables, while a parameter valuation γ is an assignment to the U variables. Sets of states are expressed as formulae in X; sets of parameter valuations are expressed as formulae in U, with each satisfying assignment corresponding to a parameter valuation. Boolean connectives have the usual set theoretical connotation (e.g., complementation is represented by logical negation, and intersection by conjunctions), while projection is represented by existential quantification.

We write $\textsc{Reachable}_S(U, X)$ for the set of reachable states in S, where a state is a valuation to the variables X and the parameters U. We notice that $\textsc{Reachable}_S(X) \wedge \gamma = \textsc{Reachable}_{S_\gamma}(X)$, i.e., the reachable state space of a parameterized system S can be seen as an association between an parameter valuation γ and the set of reachable states in the corresponding (non-parameterized) transition system S_γ.

The techniques described in this paper apply both to finite-state and to infinite-state systems. In the case of finite-state systems, termination is guaranteed; in the infinite case, convergence depends on the termination of the calls to the underlying model checking engine.

Let a property $\varphi(X)$ (φ for short) be a formula describing a set of states. A transition system S satisfies φ ($S \models \varphi$) iff $\textsc{Reachable}_S(X) \subseteq \varphi(X)$. The set of parameter valuations valid for φ for a parametric transition system S is defined as $\textsc{ValidPars}_{S,\varphi}(U) = \{\gamma \in \Gamma \mid S_\gamma \models \varphi\}$. A valid parameter valuation γ guarantees that the property φ holds in S_γ.

We consider cost functions $\textsc{Cost} : \Gamma \to \mathbb{N}$ as integer-valued functions over parameter valuations. A multi-dimensional cost function is defined as a vector of cost functions; for brevity we write $\textsc{Cost} : \Gamma \to \mathbb{N}^m$. We call $\mathbb{N}^m$ the space of costs. Notice that a cost function can be symbolically represented as a term. Given two cost vectors $(v_1, \cdots, v_m)$ and $(w_1, \cdots, w_m)$, we define the partial order relation $\preceq$ as $(v_1, \cdots, v_m) \preceq (w_1, \cdots, w_m)$ iff $\forall i.\ v_i \leq w_i$.

Given S, φ and $\textsc{Cost}$, we say that an assignment $\gamma \in \Gamma$ is Pareto-Optimal iff:

1) $S_\gamma \models \varphi$, and
2) if there is γ' s.t. $S_{\gamma'} \models \varphi$ and $\textsc{Cost}(\gamma') \preceq \textsc{Cost}(\gamma)$ then $\gamma = \gamma'$.

Pareto-optimality boils down to optimality with respect to a single cost function when $m = 1$. The cost function can be represented symbolically as a term $\textsc{Cost}(U)$; a set of assignments is then simply identified by a formula $\textsc{Cost}(U) \bowtie v$ where v is a natural number and $\bowtie$ is a relation operator.

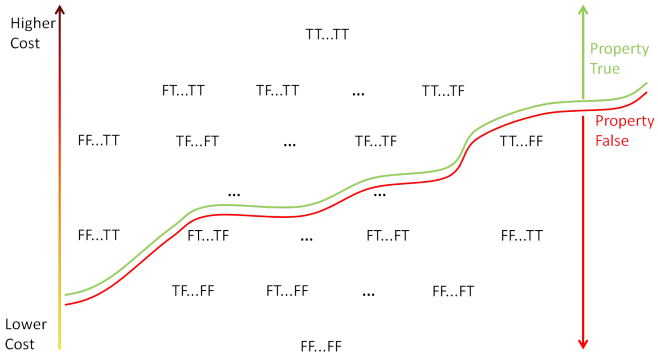

Fig. 1. Monotonicity with respect to Property and Cost function

In this paper, we make two assumptions of *monotonicity*. The first one is monotonicity of the "property holds" relation, and the second is monotonicity of the cost function.

We say that $S \models \varphi$ is monotonic w.r.t. Γ iff

$$\forall \gamma,\ \text{If } S_\gamma \not\models \varphi \text{ then } \forall \gamma'.\ \gamma' \preceq \gamma \Rightarrow S_{\gamma'} \not\models \varphi$$

Intuitively, if the property does not hold under a given parameter valuation, then it does not hold for any of its predecessors. Conversely, if the property holds under a given valuation, then it also holds for all the successors.

We say that $\textsc{Cost}$ is monotonic w.r.t. Γ iff

$$\forall \gamma, \gamma'.\ \text{If } \gamma \preceq \gamma' \text{ then } \textsc{Cost}(\gamma) \preceq \textsc{Cost}(\gamma')$$

The Pareto-Frontier $PF(\textsc{Cost}, \varphi) \subseteq \Gamma$ is the set of parameter assignments that are valid for φ and that are Pareto-optimal with respect to $\textsc{Cost}$.

The space explored in this paper is depicted in Figure 1. Above the line are the valuations that are valid for φ; below the line are the ones for which the property does not hold, i.e., the ones which are associated to at least one counterexample trace. The vertical arrow on the left denotes a cost function that is upwards monotonic along each path.

IV. Solution Description

In the following we present several algorithms for the computation of the Pareto frontier, for a given S, φ, and $\textsc{Cost}$. We assume that both the property satisfaction relation ($S \models \varphi$) and $\textsc{Cost}$ are monotonic with respect to Γ. For the sake of presentation, we assume also that there is at least one parameter valuation γ s.t. $S_\gamma \models \varphi$.

The algorithms that we present define a spectrum based on how much of the set of $\textsc{ValidPars}$ we compute up-front. In Section IV-A, we compute the whole set of good valuations up-front, and then proceed to the computation of the Pareto-Frontier. In Section IV-B, we "slice" the space $\textsc{ValidPars}$ by one dimension and compute one of the slices at the time; once a slice has been computed, we minimize w.r.t. to the other costs. By doing so, we are able to skip slices (using the monotonicity assumption), so that we end up computing a subset of $\textsc{ValidPars}$. Finally, in Section IV-C we do not compute $\textsc{ValidPars}$ directly, but navigate through

the valuations lattice driven by the cost functions and test on-the-fly membership of points to VALIDPARS.

For the sake of presentation, we describe most algorithms assuming that we have a two-dimensional cost function.

A. The valuations-first approach

function VALUATIONSFIRST(S, COST, φ)
  $VP :=$ VALIDPARS(S, φ)
  return PARETOFRONT(COST, VP)
end function

Fig. 2. Valuations First pseudo-code

function VALIDPARS(S, φ)
  $Bad := \bot$
  $S = (U, X, I, T)$
  while $S \not\models \varphi$ **do**
    $\gamma' :=$ project counter-example on U
    $Bad := Bad \vee \gamma'$
    $I := I \wedge \neg Bad$
  end while
  return $\neg Bad$
end function

Fig. 3. VALIDPARS computation

The first algorithm we present is an eager, two-stage approach. Figure 2 provides a high-level description of the algorithm.

The first stage constructs the set of parameter valuations that are valid for the φ property. This gives a closed-form representation of the solution space, regardless of cost considerations, represented as a formula VALIDPARS. The second phase carries out an analysis of the solution space taking the costs into account, selecting the assignments that are Pareto-optimal, thus building the Pareto front.

Each of the phases can be in turn refined. The computation of VALIDPARS can be carried out directly, by performing a reachability analysis on S, thus obtaining REACHABLE(U, X), and then considering only the valuations for which the states always satisfy the property. This idea has been explored with a BDD-based implementation in [13], where it was applied in the computation of Fault-Trees. In many cases, however, the computation of the reachable states can be over-killing. In Figure 3 an alternative approach is presented, based on the algorithm proposed in [14], that constructs the set VALIDPARS $= \{\gamma_i \mid S \models \gamma_i \rightarrow \varphi\}$ of valid parameter valuations. The idea is to rely on a model-checking routine to compute the set $InvalidPars = \{\gamma_i \mid S \not\models \gamma_i \rightarrow \varphi\}$, i.e., a representation of the "lower part" of the lattice in Figure 1. At a very high level, this is done by enumerating counterexample traces to the negation of φ. Each trace is associated with an invalid parameter valuation, which is then accumulated in the result, and removed from the initial states, thus preventing the model checker from re-discovering it. Once $InvalidPars$ is computed, the space of valid parameter valuations is simply obtained by complement. This algorithm can thus rely on a model-checker as a black-box, therefore leveraging recent advancements in SAT-based model-checking techniques (e.g., IC3).

The second phase carries out the optimization of the combinatorial space defined by VALIDPARS with respect to COST. This can be done, for example, by enumerating all the elements in VALIDPARS and comparing the associated costs, or by considering the symbolical characterization of the Pareto front:

$$\text{PARETOFRONT}(U) = VP(U) \wedge \nexists U'.((U' \prec_{\text{COST}} U) \wedge VP(U'))$$

A simple way of computing PARETOFRONT(U) is given by the possibility of encoding the cost functions into SAT (e.g., using SMT over bit-vectors), and applying an iterative approach that tightens the constraints on the cost along each dimension.

There are two big disadvantages in the valuations-first algorithm. First, in order to compute VALIDPARS, we need to enumerate all the elements of $InvalidPars$. This means that the first phase may be in some cases inherently expensive. Second, the first phase proceeds by under-approximating the complement of the valid space, regardless of the cost information. This means that virtually no information on what is a valid (let alone optimal) solution is found until convergence in the accumulation has been reached, i.e., until the whole $InvalidPars$ set has been computed.

B. The one-cost slicing approach

function SLICING(S, COST, φ)
  PF := $\emptyset$
  $\gamma = \top$;
  $c_1 :=$ COST$_1(\gamma)$
  $S' := FixCost(S, \text{COST}_1 = c_1)$
  $VP_{\text{COST}_1} :=$ VALIDPARS(S', φ)
  while $VP_{\text{COST}_1} \neq \emptyset$ **do**
    (γ, c_2) = MINIMIZE (COST$_2$, VP_{COST_1})
    (γ, c_1) := REDUCE $_{\text{COST}_1}(S, \gamma, \varphi, c_2)$
    PF.add(γ, c_1, c_2)
    $c_1 := c_1 - 1$
    $S' :=$ FIXCOST$(S, \text{COST}_1 = c_1)$
    $VP_{\text{COST}_1} :=$ VALIDPARS(S', φ)
  end while
  return PF
end function

function FIXCOST(S, $CostExpr$))
  $S = (U, X, I, T)$
  $S' := (U, X, I \wedge CostExpr, T)$ **return** S'
end function

Fig. 4. Slicing algorithm

The second algorithm (Figure 4) we propose interleaves the analysis of the cost information with checks on the validity of the parameters. This is done by slicing the space of valid parameters along the different dimensions (i.e., cost functions).

We first initialize c_1 to the highest possible value, and the Pareto frontier to the empty set. We iterate as follows. First, we compute all the candidate solutions on the parametric system S' in which we fixed the cost COST$_1$ to the value c_1. We then search in the set of candidates (VP_{COST_1}) for the best valuation and cost for COST$_2$. Once an optimal cost c_2 has been found, we fix it and try to find a smaller valuation w.r.t. COST$_1$, and add the solution to the Pareto front. This is done

by calling a function $\text{REDUCE}_{\text{COST}_1}$ which, given a solution γ of cost (c_1, c_2), returns another solution γ' of cost (c_1', c_2) with $c_1' \leq c_1$. For now, REDUCE is simply a function that tries to improve a candidate solution γ. We shall describe its actual implementation in the next Section.

In order to find the other points of the Pareto frontier, we decrease c_1 and test whether any solution (independently of COST_2) exists. If it does, we repeat the process, otherwise we terminate.

Note that in the function MINIMIZE we operate on the set of the solutions, while in REDUCE, we generate a candidate $\gamma' \preceq \gamma$ and test whether it is still a solution (i.e. $S_{\gamma'} \models \varphi$). Due to the monotonicity assumption, REDUCE cannot skip solutions. However, REDUCE can drastically accelerate the search by avoiding the enumeration of all costs in c_1.

In the pseudo-code, the addition of solutions to the Pareto front is slightly simplified. In practice, we cannot add a solution γ_1 immediately in the Pareto front, but we need to wait for the next solution γ_2 to be added ($PF.add$). If the costs of γ_1 and γ_2 are incomparable, then γ_1 is Pareto-optimal and gets added to the frontier. If γ_2 is smaller than γ_1, then γ_1 is not optimal and is discarded. This pair-wise operation guarantees that only Pareto optimal solutions are considered.

This approach only computes slices of VALIDPARS when needed. Although in the worst-case we can end-up computing it all by slices, when calling the REDUCE function, it is generally possible to accelerate the search and skip intermediate slices.

C. The costs-first approach

```
function CostsFirst(S, Cost, φ)
    PF := ∅
    γ := ⊤;
    c1 = Cost1(γ); c̄2 = Cost2(γ)
    repeat
        c2 = c̄2
        for γi ∈ MaxSmallerCandidate_Cost2(c1, c2) do
            if S_γi ⊨ φ then
                (γ, c2) := Reduce_Cost2(S, γ, φ, c1)
            end if
        end for
        (γ, c1) := Reduce_Cost1(S, γ, φ, c2)
        PF.add(γ, c1, c2)
        c1 := c1 − 1
    until No solution exists for FixCost(S, Cost1 = c1)
    return PF
end function
```

Fig. 5. CostsFirst pseudo-code

One of the key insights of the slicing algorithm is that big parts of VALIDPARS might not be necessary in order to compute the Pareto front. In the costs-first approach we take this idea to the extreme: we do no compute VALIDPARS anymore. Instead, we explore the lattice of valuations induced by the cost functions. Every time we find a promising valuation γ, we test whether it is actually a solution (i.e., $S_\gamma \models \varphi$). Due to the monotonicity assumption, whenever we find a valuation that is not a solution, we can prune all of its predecessors in the lattice (since they cannot be solutions either).

An overview of the algorithm is provided in Figure 5. We start by getting an upper-bound on both costs by considering the cost of the top valuation. In the outer-loop we decrease the value of COST_1, similarly to the slicing approach. Within the inner-loop, however, we proceed by enumerating the solutions that have smaller value w.r.t. COST_2. In particular, MAXSMALLERCANDIDATE returns the maximal solution(s) with the same cost c_1 but with smaller c_2.

The process terminates whenever no solution can be found for a given value of COST_1. Note how the structure of this algorithm is similar to the one of the slicing approach. The main difference is that we never need to compute VALIDPARS.

This algorithm allows us to find subsets of the Pareto front, since it can be interrupted at any point and is guaranteed to provide an under-approximation of the Pareto-Frontier. This is in contrast with the approaches described in Section II for parameter synthesis, that require termination of the procedure in order to provide the solution space of the parameters.

V. IC3-BASED IMPLEMENTATION

We implemented the approaches described in the previous section using IC3-based techniques.

In particular, there are two key ideas that we can leverage in order to have an efficient algorithm using IC3. First, we notice that $S_\gamma \models \varphi$ holds iff $S \models \gamma \rightarrow \varphi$. This observation makes it possible to reason always on the same system, and moves the choice of the valuations within the property. This leads us to the second fundamental observation. If $S \models \gamma \rightarrow \varphi$, we can extract from the IC3 model-checker the inductive invariant ψ. By definition of inductive invariant we know that $\psi \models \gamma \rightarrow \varphi$; moreover, it might be the case that we can reuse the same invariant to check whether another valuation γ' is a solution: i.e., $\psi \models \gamma' \rightarrow \varphi$. We will use this idea when trying to reduce the valuation, since this makes it possible to reason locally on a (relatively small) formula, and does not require unrolling or computing reachable states. The efficiency of the procedure will then largely depend on how well the reduction step works.

Figure 6 presents the adaptation of the costs-first algorithm when using the inductive invariant to perform the REDUCE step. The same idea for the REDUCE can be applied in the slicing algorithm.

We navigate the lattice by picking the maximal candidate(s) of smaller cost w.r.t. COST_2 (MAXSMALLERCANDIDATE). This fact guarantees that the algorithm will terminate, since we are always picking a solution of smaller dimension. We then check that the property still holds for the new valuation γ_i, by using IC3. If this is the case, we are provided with an inductive invariant ψ, s.t., $\psi \models \gamma_i \rightarrow \varphi$.

The operation of picking a cost-predecessor could be, in principle, delegated to a pseudo-boolean constraint solver, or to other reasoning engines that are able to deal with costs natively. For our simple implementation, we use an SMT solver with the theory of bit-vectors.

When considering the parameters as a set of elements, we can try to minimize the set by implementing the REDUCE procedure using unsat-cores. Namely, we check the unsatisfiability of $\psi \wedge \neg\varphi$ under the assumption of γ_i and use standard features

```
function CostsFirstIC3(S, Cost, φ)
    PF := ∅
    γ := ⊤;
    c1 = Cost1(γ); c̄2 = Cost2(γ)
    repeat
        c2 := c̄2
        for γi ∈ MaxSmallerCandidate_Cost2(c1, c2) do
            (res, ψ) := IC3(S, γi → φ)
            if res == Safe then
                # ψ is an inductive invariant s.t. ψ ⊨ γi → φ
                (γi, c1, c2) := Reduce_Cost2(ψ, γi, φ)
            end if
        end for
        (γi, c1, c2) := Reduce_Cost1(ψ, γi, φ)
        PF.add(γ, c1, c2)
        c1 := c1 − 1
    until No solution exists for FixCost(S, Cost1 = c1)
    return PF
end function
```

Fig. 6. IC3-based CostsFirst pseudo-code

from modern SAT solvers to minimize the unsat-core that, in turn, translates in picking a subset of the parameters that makes the formula unsatisfiable. By doing so, we are able to "jump" and quickly reduce the valuation γ. For integer parameters, instead, we use a REDUCE procedure that performs a linear or binary search, using the inductive invariant.

In general, we could use the identity function as REDUCE, and this would still guarantee the correctness and termination of the algorithm. However, this would end-up requiring an explicit state search of the lattice. Having a smart REDUCE procedure makes it possible to jump and terminate faster.

Since the inductive invariant does not depend on the costs, it is possible to reuse the invariant from previous calls in an incremental way. Intuitively, this provides us with stronger invariants that are more likely to allow us to reduce the parameters aggressively.

VI. MOTIVATING APPLICATION DOMAINS

We describe now two motivating application domains: sensor placement for diagnosability and product line engineering.

Sensor Placement: The problem is of practical relevance, and substantial interest has been devoted to it in the setting of autonomous systems. Typical architectures integrate components for Fault Detection and Identification (FDI) that are used to detect, during operation, whether some (and which) faults may have occurred [24], [10]. The information produced by FDI is then used for Fault Isolation and Recovery, i.e., to respond to the effects of faults, e.g., by reconfiguration.

Intuitively, the problem is to identify a suitable set of sensors that will allow the FDI subsystem to have enough information to carry out, within a given delay, its diagnosis task. In this setting, a parameter represents whether a sensor is present in the design, and a parameter valuation identifies a subset of all available sensors. There is a trade off between the observation power of the available sensors, and the delay required to diagnose a certain condition of interest. Intuitively, a reduction in the set of sensors may lead to an increase in the delay.

The property φ that we want to show is *diagnosability* with respect to a given delay d, i.e., the ability to detect an event of interest within at most d steps. In the case of a given set of sensors, this is reduced to the model checking problem on the so-called *twin plant*, a construction based on the composition of two replicas of the plant under observation [10]. The twin plant encodes the existence of a critical pair, i.e., two indistinguishable traces satisfying a pair of conditions of interest (e.g., the occurrence of two faults of different type that must be identified).

The problem is generalized by considering a parameter set U, defined as $\{s_1, \ldots, s_k, d\}$, where a valuation to the vector $(s_1, \ldots, s_k)$ of k sensor parameters identifies a configuration in the design space. The delay d is an integer valued parameter. The space of assignments is then $\mathbb{B}^k \times \mathbb{N}$

The diagnosability property φ is defined as the invariant property:

$$\neg((delay_\psi \geq d) \wedge obs_{eq})$$

where $delay_\psi$ counts the steps since the occurrence of the condition of interest ψ. This formula is satisfied in the twin-plant iff there is no critical pair. In general, we assume that an upper-bound on the delay for the model is known. In addition to the theoretical bound that always exists for state transition systems, in practice there usually is an application specific time-frame after which the diagnosis is not useful anymore (e.g., mission life-span, propagation time). Several interesting COST functions are possible. For the sensor placement problem we use one based on weights/delay pairs:

$$\text{Cost}_{weighted}(s_1, \ldots, s_k, d) = (\sum_i w_i s_i,\ d)$$

Product Lines: When designing a product, engineers are often faced with a high degree of variability in terms of possible features. Such variability is usually captured in product line models (sometimes referred to as feature models). For instance, in [16] the authors model variability in a controller design, and the authors of [6] consider software product lines. Here we are specifically interested in the analysis of dynamic systems as opposed to static contexts which are usually addressed with constraint programming techniques.

The goal of product line engineering is usually to identify which combinations of features satisfy a certain property. Here however we specifically address the Pareto-optimal trade-off problem. In various works there are different assumptions on the monotonicity of features, that is whether by adding features the possible behaviors increase monotonically or whether some behaviors can be overridden. In our work we only assume the monotonicity of the property of interest in terms of feature additions.

VII. EXPERIMENTAL EVALUATION

A. Experimental Setup

We have implemented the algorithms described above on top of the NUXMV model checker [25]. Although our framework can in principle support any number of cost functions, our current implementation only supports two of them. The executables and benchmark instances used in the evaluation

are available at https://es-static.fbk.eu/people/griggio/papers/fmcad14pareto.tar.bz2.

We evaluate our approach on a series of benchmarks coming from the domains of sensor placement [10] and product lines [16]. The sensor placement examples were obtained from realistic case studies on a simpler problem, i.e. finding a good set of sensors for a fixed delay. These simpler benchmarks were challenging and in some case not solvable for our previous techniques [10]. The properties for the product lines benchmarks that were derived for our invariant checking framework are artificial but tailored specifically for these examples. For both cases we are unfortunately not aware of other publicly available industrial benchmarks.

ORBITER, ROVERSMALL, and ROVERBIG are models of an orbiter and of a planetary rover, both developed in the OMCARE project (see [26], [27]). The models describe the functional level, with various relevant subsystems including failure modes. CASSINI models the propulsion system of the Cassini spacecraft (see [13]). It is composed of two engines fed by redundant propellant/gas circuit lines, which contain several valves and pyro-valves. Leakage failures are attached to all components. ELEVATOR models an elevator controller, parameterized by the number of floors. The modeled properties are cabin and door movement, request and reset operations at each floor, and the controller logic. C432 is a boolean circuit used as a benchmark in the DX Competition [28], whose gates can permanently fail (inverted output). The observables are the inputs and output values for the gates of the circuit. X34 is a benchmark describing a simplified version of the main propulsion system of a spacecraft [29]. All models also contain faults based on which a sensor placement problem is formulated. PRODUCT LINES are benchmark instances derived from [16], describing a railway switch controller. The product-line features correspond to possible communication strategies used by the controller. We explore a design trade-off along two dimensions. The first is the upper bound on message sequence lengths. The second one is a cost associated to dropping a particular feature, specified by a random cost function. Our aim is to minimize both the message sequence bound and the cost of removing features in order to guarantee it.

For each example, we generated multiple benchmarks by varying both the number of parameters considered and the (randomly-generated) costs of the individual parameters. Overall, our benchmark set consists of 81 instances. The number of Pareto-optimal solutions varies between 0 and 5.

B. Results

Family	#Instances	valuations-first	one-cost slicing	costs-first
c432	32	11	13	**32**
cassini	21	6	12	**21**
elevator	4	4	4	4
orbiter	4	4	4	4
roversmall	4	4	4	4
roverbig	4	4	4	4
x34	4	4	4	4
product lines	8	6	4	**8**
TOTAL	81	43	49	**81**

Fig. 7. Number of solved instances by each approach

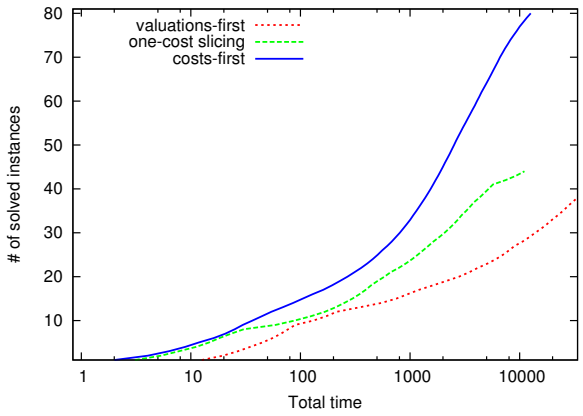

Fig. 8. Accumulated-time plot showing the number of solved instances (x-axis) in a given total time (y-axis) for the various algorithms.

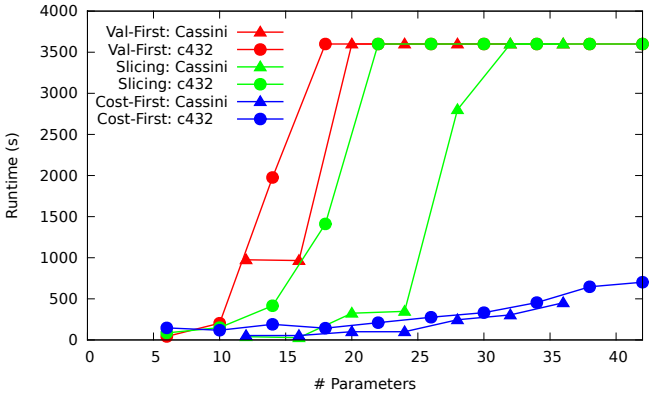

Fig. 9. Runtime for different number of parameters

We executed the experiments on a Linux cluster equipped with 2.5Ghz Intel Xeon CPUs with 96Gb of RAM. We used a time limit of 1 hour and a memory limit of 6Gb.

In Figure 7 we present the number of instances solved for each problem family. For the C432, CASSINI and PRODUCT LINES benchmarks, we can see how the costs-first approach finds all the solutions within the timeout, whereas the other two approaches fail on several instances. Figure 8 shows the accumulated-time plots for the different algorithms, plotting the number of solved instances (y-axis) in the given total amount of time (x-axis) in logarithmic scale.

For the C432 and CASSINI benchmark, we show in Figure 9 the runtime as a function of the parameters. As expected, on the same model, the number of parameters has a big impact on the runtime. Indeed, for the valuations-first and the one-cost slicing approaches this has an exponential tendency.

Finally, in order to evaluate the impact of the REDUCE procedure in the costs-first algorithm, we performed an experiment in which we ran costs-first without applying REDUCE. From the scatter plot of Figure 10, we can see that REDUCE is crucial for performance: without it, costs-first solves only 48 instances, and the runtimes increase by orders of magnitude.

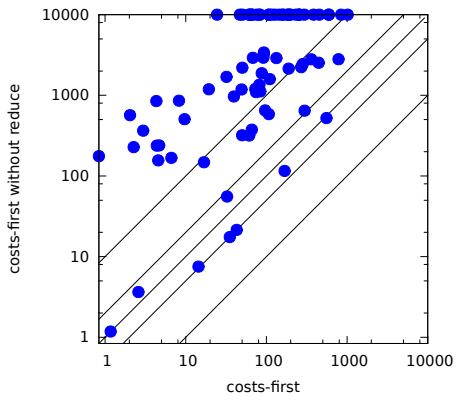

Fig. 10. Impact of REDUCE in the costs-first algorithm.

VIII. CONCLUSIONS

In this paper we have proposed a new method for the synthesis of optimal parameter valuations for multi-dimensional monotonic cost functions, enabling the construction of the Pareto front with respect to the cost function.

We analyzed three algorithms of increasing efficiency, that interleave in various ways the search for parameter valuations that satisfy the property and the optimization with respect to costs, and we showed how to implement them on top of IC3, exploiting its features for efficiency.

Our experimental evaluation shows the feasibility of the approach on benchmarks from important practical domains, and demonstrates the importance of a tight integration between model checking and cost optimization.

In the future we will generalize the approach to deal with real-valued parameters; in particular, we will investigate the generalization of the notion of monotonicity. From a practical point of view, it would be important to find an effective way of automatically testing the monotonicity assumptions. We will also generalize our implementation to support more than two cost functions, and devise strategies to handle multiple cost functions in an effective way. Further interesting directions are the investigation of specialized techniques for specific patterns of properties (e.g., response time), thus enabling the approach to be applied beyond safety properties, and techniques for relaxing the assumptions of monotonicity currently required.

REFERENCES

[1] P. Manolios, D. Vroon, and G. Subramanian, "Automating component-based system assembly," in *ISSTA*, 2007, pp. 61–72.

[2] C. Hang, P. Manolios, and V. Papavasileiou, "Synthesizing cyber-physical architectural models with real-time constraints," in *CAV*, 2011, pp. 441–456.

[3] W. Steiner and B. Dutertre, "Layered diagnosis and clock-rate correction for the ttethernet clock synchronization protocol," in *PRDC*, 2011, pp. 244–253.

[4] S. Samii, P. Eles, Z. Peng, and A. Cervin, "Design optimization and synthesis of flexray parameters for embedded control applications," in *DELTA*, 2011, pp. 66–71.

[5] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, "Constraint-driven synthesis and tool-support for flexray-based automotive control systems," in *CODES+ISSS*, 2011, pp. 139–148.

[6] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," in *ICSE*, 2011, pp. 321–330.

[7] S. Mamagkakis, D. Atienza, C. Poucet, F. Catthoor, D. Soudris, and J. Mendias, "Automated exploration of pareto-optimal configurations in parameterized dynamic memory allocation for embedded systems," in *DATE*, 2006, pp. 874–875.

[8] A. Cimatti, L. Palopoli, and Y. Ramadian, "Symbolic computation of schedulability regions using parametric timed automata," in *RTSS*. IEEE Computer Society, 2008.

[9] A. Grastien, "Symbolic testing of diagnosability," in *Twentieth International Workshop on Principles of Diagnosis (DX-09)*, 2009.

[10] B. Bittner, M. Bozzano, A. Cimatti, and X. Olive, "Symbolic Synthesis of Observability Requirements for Diagnosability," in *AAAI*, 2012.

[11] V. Pareto, *Manuale di economia politica*, ser. Collezione saggi & documenti. Edizioni Studio Tesi, 1994.

[12] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. M. III, and J. Railsback, "Fault tree handbook with aerospace applications," *Technical report, NASA*, 2002.

[13] M. Bozzano, A. Cimatti, and F. Tapparo, "Symbolic fault tree analysis for reactive systems," in *ATVA*. Springer, 2007, pp. 162–176.

[14] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Parameter synthesis with ic3," in *FMCAD*. IEEE, 2013, pp. 165–168.

[15] A. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, 2011, pp. 70–87.

[16] J. Greenyer, A. Sharifloo, M. Cordy, and P. Heymans, "Efficient consistency checking of scenario-based product-line specifications," in *Requirements Engineering Conference (RE)*, 2012, pp. 161–170.

[17] S. Reimer, M. Sauer, T. Schubert, and B. Becker, "Using maxbmc for pareto-optimal circuit initialization," in *DATE*, 2014, pp. 1–6.

[18] J. Legriel, C. L. Guernic, S. Cotton, and O. Maler, "Approximating the pareto front of multi-criteria optimization problems," in *TACAS*, 2010, pp. 69–83.

[19] T. A. Henzinger and P.-H. Ho, "Hytech: The cornell hybrid technology tool," in *Hybrid Systems*, 1994, pp. 265–293.

[20] F. Wang, "Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures," *IEEE Trans. Soft. Eng.*, vol. 31, no. 1, pp. 38–51, 2005.

[21] G.Frehse, S. Jha, and B. Krogh, "A counterexample-guided approach to parameter synthesis for linear hybrid automata," in *HSCC*, 2008, pp. 187–200.

[22] É. André, L. Fribourg, U. Kühne, and R. Soulat, "IMITATOR 2.5: A tool for analyzing robustness in scheduling problems," in *FM*, 2012, pp. 33–36.

[23] É. André and U. Kühne, "Parametric analysis of hybrid systems using HyMITATOR," in *iFM*, 2012, pp. 16–19.

[24] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta, "Formal design of fault detection and identification components using temporal epistemic logic," in *TACAS*. Springer, 2014, pp. 326–340.

[25] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The NUXMV symbolic model checker," in *CAV*, ser. LNCS. Springer, 2014.

[26] M. Bozzano, A. Cimatti, A. Guiotto, A. Martelli, M. Roveri, A. Tchaltsev, and Y. Yushtein, "On-board autonomy via symbolic model-based reasoning," in *Proceedings of the 10th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, 2008.

[27] M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev, "A comprehensive approach to on-board autonomy verification and validation," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

[28] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, and A. van Gemund, "Empirical evaluation of diagnostic algorithm performance using a generic framework," *International Journal of Prognostics and Health Management*, Sep 2010.

[29] A. Cimatti, C. Pecheur, and R. Cavada, "Formal verification of diagnosability via symbolic model checking," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.