Shorter Reasoning About Larger Requirements Models

George Mathew, Tim Menzies
Computer Science, NC State University, USA
george.meg91,tim.menzies@gmail.com

Neil A. Ernst, John Klein
Software Engineering Institute, Pittsburgh, USA
nernst,jklein@sei.cmu.edu

Abstract—Requirements models support communication and decision-making. However, when requirements models get too complex, it becomes difficult for stakeholders to understand all their nuances. Empirical and theoretical results show that automatically reasoning about complex RE models using standard methods takes exponential time that cripples the ability of these models to support decision making for large models.

One resolution to this problem comes from AI where researchers report that many models have "keys". These are model elements—decisions, in RE models—with the property that once values are assigned to the keys, it becomes very fast to reason over the remaining decisions. Using a toolkit we call SHORT, this paper seeks, and finds, such keys in eight large RE models. The number of keys were typically only 12% of all decisions. Using those keys, when optimizing to achieve the most goals at least cost, SHORT ran 100 to 1000 times faster than standard methods.

Our conclusion is that that there is much value in hunting for "keys" in RE models. Finding those keys is not complex: SHORT runs in low order polynomial time and terminates in just a few minutes (even for our largest models). Also, when found, model-based RE can be greatly SHORTened by focusing stakeholder discussion on just the key decisions.

Index Terms—Requirements engineering, softgoals, optimization, search-based software engineering.

I. INTRODUCTION

When reasoning about complex requirements, it can be useful to build a model that documents the beliefs of all the stakeholders. Many researchers [1, 2, 3, 4, 5, 6, 7, 8] report that the process of building and analyzing such a model can help stakeholders better understand the ramifications of their decisions.

That said, complex models can sometimes overwhelm stakeholders. Consider a committee reviewing the goal model shown in Fig. 1 that describes the information needs of a computer science department [9]. Such a committee may have trouble with manually reasoning about all the conflicting relationships in this model. Further, automatic methods for reasoning about these models are hard to scale up: as discussed below, reasoning about inference over these models is an NP-hard task.

But are models like Fig. 1 as complex as they appear? That graph is somewhat of a tangle—if we straightened out all the dependencies, would we find that much of this model depends on just a few *key* decisions? As discussed below, such keys have been reported in other domains [10, 11, 12, 13]. The importance of key decisions is that, once their values are assigned, it becomes very simple to reason over the remaining decisions.

This paper argues that it is beneficial to look for these keys in requirements models since, if they exist, we can achieve "shorter" reasoning about large RE models, where "shorter" is measured as follows:

- Large models can be processed in a very short time;
- Runtimes for automatic reasoning about RE models is shorter so stakeholders can get faster feedback from their models;
- The time required for manual reasoning about models is shorter since stakeholders need only debate a small percent of the issues (just the key decisions).

This paper is structured as follows. The next section presents a small RE model where dozens of decisions are controlled by just three keys. The generality of that observation is then discussed. This small example will be shown to belong to a class of models that appear widely in the RE literature. Also, as discussed in the AI literature, many researchers report that many models have keys. After that, we introduce SHORT, a novel toolkit for finding and using keys. SHORT is then applied to eight RE models from the i* community [6] (see goo.gl/K7N6PE). When translated into a node-arc format, the models have the sizes shown in Table I.

From those experiments, we will conclude:

1) *Keys can be found in numerous RE models.* In the sample of models explored here, the number of keys per model is small: typically 12% of all decisions.
2) *Keys are easy to find.* SHORT runs in low-order polynomial time and terminates in just a few minutes (even for our largest model). This is a significant result: all prior attempts suffered from crippling runtimes that prevented scale up [14, 15].
3) *It is very useful to apply keys for reasoning about RE models.* Using the keys, SHORT can find decisions that lead to most goals at least costs *100s to 1000s of times faster than standard methods.* Further, the decisions found by SHORT are competitive, and sometimes even better than those found by standard optimizers.
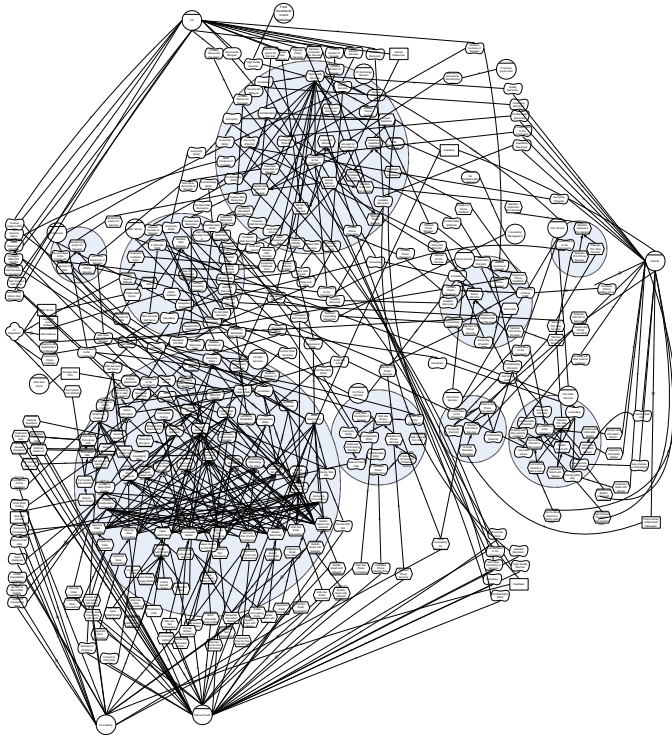
Fig. 1. A goal model presenting choices for services within a CS department. Gray circles are stakeholder specific issues. If the reader finds the complexity of this model confusing, then our point is made.

II. An Introductory Example of Key Decisions

Fig. 2 shows a requirements model about IT System modernization used by the authors at the Software Engineering Institute. Examples of modernizations include the Y2K problem (moving from 2 digit to 4 digit years); end-of-life decisions on operating systems and database products; improving architectural support for new capabilities (e.g. support for mobile devices); etc This model comments on the refactoring, re-architecting, and redesign of existing systems. The model has the syntax of Fig. 3.

Note that the model has dozens of decisions and a few top-level goals shown circled top right: "Good Example ...", "Easily Share Data Internally", "Modernize". All these nodes have some cost which can be set using stakeholder knowledge. If the stakeholders are not sure of the exact costs, we would let them define a range then conduct Monte Carlo simulations that sample randomly across that range. That said, for the rest of this paper we assume a decision to use any node costs the same fixed amount (and in our modeling language, this is a simple facet to change).

The two red and one green decisions in Fig. 2 are the keys to this model. To see this, then consider:

- The right-hand-side of this model has not connection to the high-level goals. Hence, two sensible decisions for this model would be to deny the leaves marked in dark red, thus disabling the right tree.

Model	Nodes	Edges	
Services	351	510	(see Fig. 1)
Counselling	350	470	
Marketing	326	422	
Management	206	239	
SAProgram	114	168	
ITDepartment	126	162	
Kids&Youth	81	81	
IT Modernization	53	57	(see Fig. 2)

- As to the left-hand-side of the model, the leaf marked in dark green ("J2EE specification") selects the shortest left-hand-side sub-branch with fewest leaves. If the goal is achieve the most high-level goals at least cost, then it is also sensible to take this single dark green. Why? Well, making any other decision one the left-hand-side model will significantly add to the overall cost of the solution, since all those other decisions require multiple conjunctions of decisions.

III. Frequently Asked Questions

When discussing these results with our colleagues, there are several frequently asked questions.

Question 1: What if users reject the analysis of the last section and start exploring the deeper left-hand-side branches?

This would tell a requirements engineer that minimizing cost is *not* a primary goal of this software team. In that case, the requirements engineering would rerun SHORT, but remove the "minimizing cost" goal. That would lead to new alternatives which can be debated by the stakeholders. Note that it would not be a burdensome task to run SHORT a few extra times since, as shown below, SHORT runs in just a few seconds.

Question 2: All your experiments use the i models shown in Table I. So are your results really just some quirk of that modeling language?*

We use the models of Table I for two reasons. Firstly, they are the largest requirements models we can access. Secondly, these i* models are representative of a large class of other requirements languages; i.e. those that can be compiled into what we call PCBN; i.e. *Propositional assertions*, where subsets of the propositions are augmented with with *Cost*s, *Benefit*s, and *Nogood*s:

- The use of *Cost*s was discussed, earlier in this page.
- *Benefit*s relate to the high-level goals of the system. For example, in Fig. 2, the lower-level leaves have zero benefit while the goals circled top-left have substantial benefit.
- *Nogood*s report what variable assignments are not allowed. For example, if a variable can take multiple mutual exclusive values, then it would be *nogood* for that variable to be given two different values.

Many modeling notations can can be converted into PCBN including the i* notation of Fig. 2 [5]. For example, Menzies'
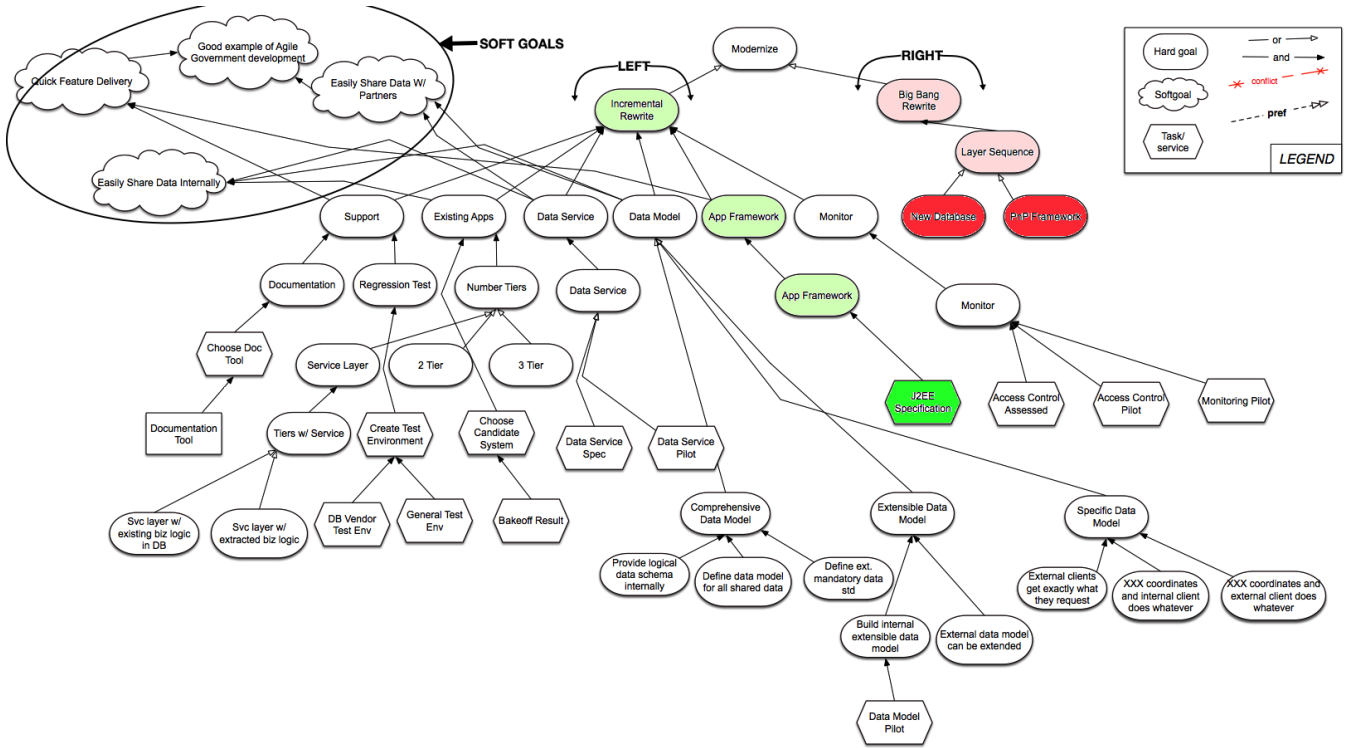
Fig. 2. A goal model for IT System modernization. This model has the syntax of Fig. 3. The three keys of this model are the leaf nodes shown in red and green (see text for an explanation of why these are the keys).

1. **Nodes:** have labels and *decisions* are label assignments.
2. **Edges:** *Con* nodes connect to other nodes via one four edges types "makes, helps, hurts, breaks" with weights $E_j \in \{1, \frac{1}{2}, -\frac{1}{2}, -1\}$, respectively.
3. **Node types:** Nodes in a goal model can be of type *leaf*, *com*bine, or *con*tribute (*com* nodes have sub-types *and, or*). *Leaf* nodes are different to the rest since they have no children. On the other hand, when dealing with *and,or* nodes, it is expected to meet *all,one* (respectively) of the requirements in the child nodes. *Con* nodes divide into *softgoals*, which users are willing to surrender if need be, and *hardgoals* which users are more eager to achieve.
4. **Labels:** Nodes N_i have labels $\{1, \frac{1}{2}, 0, -\frac{1}{2}, -1\}$ for "satisfied, partially satisfied, undefined, denied, partially denied".
5. Initially, labels are *undefined* and are then relabelled *satisfied* or *denied* by some labeling procedure (e.g. see Fig. 5) during which labels may temporarily labeled *partially satisfied/denied*.

Fig. 3. Syntax of goal models.

HT4 system included a domain general *knowledge compiler* that generated PCBN from (a) causal diagrams; (b) qualitative equations; a (c) frame-based knowledge representation; (d) compartmental models, and even (e) first-order systems (with finite-domains on the variables) [16].

Further, there any many recent examples of such knowledge compilation in the RE community. For example, any RE researcher using SAT solvers must write a knowledge compiler to convert their high-level notations into PCBN.

Some of that research comes from within the i* community (e.g. [17]) and some from elsewhere. For example, see researchers using SAT solvers to explore van Lamsweerde's goal graphs [18]; requirements for software product lines and feature models [19, 20], as well as other RE tools [21]. For a discussion on other RE models that can be compiled to propositions, see [5].

Q3: This paper proposes a new softgoal execution tool, yet many other such tools exist. Why propose a new one?

As mentioned above, many researchers use SAT solvers to find a solution within softgoals [22]. For example, Horkoff and Yu, in [23] discuss local propagation methods for reasoning backwards from goals or forwards from assertions across models like Fig. 2 (and sometimes, the Horkoff and Yu methods call SAT solvers as a sub-routine). Using SMT provers, these SAT solver methods can handle 1000s of elements (see the recent work of Nguyen *et al.*[24]).

Where SHORT differs from the above is that those methods find *one* solution while SHORT tries to build a trade space that summarizes the effects of decisions across *all* solutions. As discussed later, this *all* solution approach is NP-hard and, prior to this paper, all attempts to address this problem have suffered from crippling runtimes that that prevented scale up [11, 12]. What makes SHORT interesting is it is the first implementation we have found (in the last 20 years) that achieves low-order polynomial runtimes for large RE models.

3

That said, there could be advantages to *combining* SHORT with a forward or backwards analysis:

- Since SHORT is so fast, it could be run prior to a forward/backwards analysis in order to generate the trade space.
- Stakeholders could then use it as a guide while debating trade-offs during the forward/backwards analysis.

This could be one very promising future direction for this research.

Question 4: You have shown in Fig. 2 that this model has three keys. But what does that say about other RE models? Are these keys are just a quirk of this model?

As discussed in the next section, there is much evidence suggests that the results from Fig. 2 will generalize to many more models. In the field of AI, Many other research areas have reported that models often contain a small number of key decisions (see discussion in §IV-A); and our own work has shown similar findings in software-specific areas—see [25, 26, 27]. Further, as shown in the experiments at the end of this paper, these keys are common in RE models.

IV. RELATED WORK

A. Complexity of Processing RE Models

Given multiple stakeholders writing assertions into a requirements model, it is likely that those stakeholders will generate models that can contain contradictions; i.e. incompatible assignments of labels to variables. For example, in Fig. 2, it would be a contradiction to assign *satisfied* and *denied* to the same variable.

In traditional logic, if some set of assertions generates a contradiction then all the assertions are discarded as inconsistent. But in requirements models, when inconsistencies are detected, it is standard practice to focus on zones of agreement and avoid the parts of the model leading to the inconsistencies. Nuseibeh lists several strategies for this approach [28]:

- *Ignore:* Skip over edges that lead to contradictions;
- *Circumventing:* "Slip around" inconsistencies; i.e. if inference is blocked due to inconsistency, the inference can explore other avenues.
- *Ameliorating:* When conflicts cannot be avoided, it is prudent to try reduce the total number of conflicts.

The problem with these tactics is that they can be very slow. Formally, the study of models with contradictions is the called "abductive reasoning". Poole's THEORIST system [29] offers a clean logical framework of such reasoning. In that framework, a goal graph is a *theory* that contains a small number of upper-most *goals*. When we reason about that theory, we make *assumptions* about either (a) initial facts or (b) how to resolve contradictory decisions. In the general case, only some subset of the theory can be used to achieve some of the goals using some of the assumptions without leading to contradictions (denoted $\perp$). That is:

$$
\begin{aligned}
T &\subseteq theory \\
A &\subseteq assumptions \\
G &\subseteq goals \\
T \wedge A &\vdash G \\
T \wedge A &\nvdash \perp
\end{aligned}
\tag{1}
$$

A *world of belief* is a solution that satisfies these invariants. For many years we have tried to find such worlds using a variety of methods. For example Menzies' HT4 system [14] combined forward and backward chaining to generate worlds while Ernst et al. [15] used DeKleer's ATMS (assumption-based truth-maintenance system) [30]. Those implementations suffered from cripplingly slow runtimes that scaled very poorly to larger models. Such slow runtimes are not merely a quirk of those implementations—rather they are fundamental to the process of exploring models with many contradictions. It is easy to see why: exploring all the the subsets in Equation 1 is a very slow process. Bylander et al. [31] and Abdelbar *et al.* [32] both confirm that abduction is NP-hard; i.e. when exploring all options within a requirements goal model, we should expect very slow runtimes.

B. Reports of "Keys" in AI Researcch

Just because an inference task is NP-hard, that does not necessarily mean that that task will always exhibit exponential runtimes. Numerous AI researchers studying supposedly NP-hard tasks report the existence of a small number *key* variables that determine the behavior of the rest of the model. When such keys are present, then the problem of controlling an entire model simplifies to just the problem of controlling the keys.

Keys have been discovered in AI many times and called many things: *variable subset selection, narrows, master variables*, and *backdoors.* In the 1960s, Amarel observed that search problems contain *narrows*; i.e. tiny sets of variable settings that must be used in any solution [11]. Amarel's work defined macros that encode paths between the narrows in the search space, effectively permitting a search engine to leap quickly from narrows to narrows.

In later work, data mining researchers in the 1990s explored and examined what happens when a data miner deliberately ignores some of the variables in the training data. Kohavi and John report trials of data sets where up to 80% of the variables can be ignored without degrading classification accuracy [13]. Note the similarity with Amarel's work: it is more important to reason about a small set of important variables than about all the variables.

At the same time, researchers in constraint satisfaction found "random search with retries" was a surprisingly effective strategy. Crawford and Baker reported that such a search took less time than a complete search to find more solutions using just a small number of retries [12]. Their ISAMP "iterative sampling" algorithm makes random choices within a model until it gets "stuck"; i.e. till further choices do not satisfy expectations. When "stuck", ISAMP does not waste time fiddling with current choices (as done

Decision Structure
(Softgoal Model)

Decision Cost/Benefit
(Existing Technology)

Sample Heuristically & Optimize

What are the best combinations of decisions?

Bayesian Support
(Rank)

Which decisions appear in the best solutions?

Key Decisions
(the ones that matter)

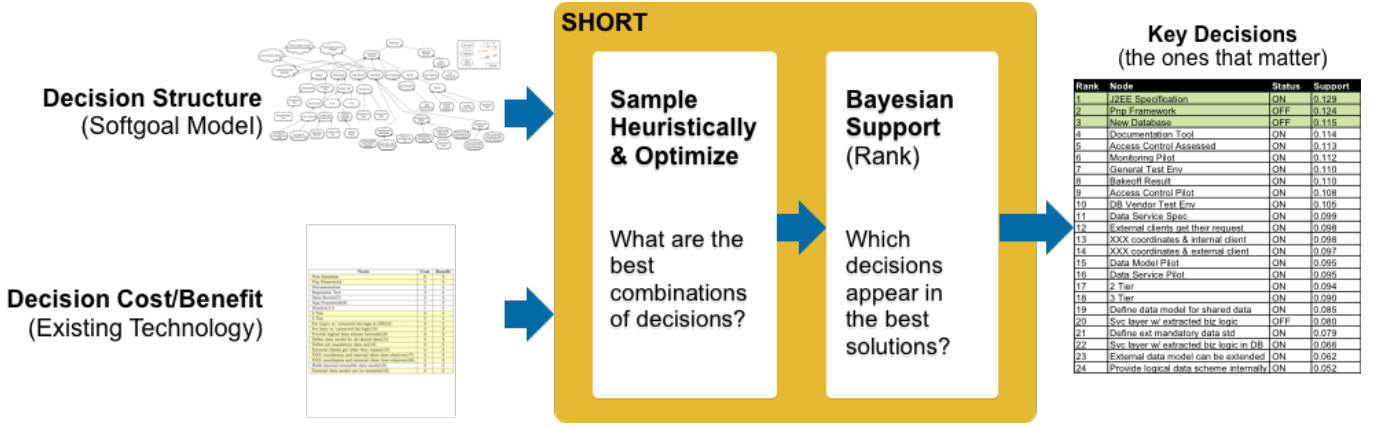| Rank	Node	Status	Support
1	J2EE Specification	ON	0.129
2	Pnp Framework	OFF	0.124
3	New Database	OFF	0.115
4	Documentation Tool	ON	0.114
5	Access Control Assessed	ON	0.113
6	Monitoring Pilot	ON	0.112
7	General Test Env	ON	0.110
8	Bakeoff Result	ON	0.110
9	Access Control Pilot	ON	0.108
10	DB Vendor Test Env	ON	0.105
11	Data Service Spec	ON	0.099
12	External clients get their request	ON	0.098
13	XXX coordinates & internal client	ON	0.098
14	XXX coordinates & external client	ON	0.097
15	Data Model Pilot	ON	0.095
16	Data Service Pilot	ON	0.095
17	2 Tier	ON	0.094
18	3 Tier	ON	0.090
19	Define data model for shared data	ON	0.085
20	Svc layer w/ extracted biz logic	OFF	0.080
21	Define ext mandatory data std	ON	0.079
22	Svc layer w/ extracted biz logic in DB	ON	0.066
23	External data model can be extended	ON	0.062
24	Provide logical data scheme internally	ON	0.052

Fig. 4. Finding solutions with SHORT. Costs and benefits are inputs are domain-specific constructs that, in our work, we develop with users as part the model elicitation process.

by older chronological backtracking algorithms). Instead, ISAMP logs what decisions were made before getting "stuck". It then performs a "retry"; i.e. resets and starts again, this time making other random choices to explore.

Crawford and Baker explain the success of this strange approach by assuming models contain a small set of *master variables* that set the remaining variables (and this paper calls such master variables the *key decisions*). Rigorously searching through all variable settings is not recommended when master variables are present, since only a small number of those settings actually matter. Further, when the master variables are spread thinly over the entire model, it makes no sense to carefully explore all parts of the model since much time will be wasted "walking" between the far-flung master variables. For such models, if the reasoning gets stuck in one region, then the best thing to do is to leap at random to some distant part of the model.

A similar conclusion comes from the work of Williams et al. [10]. They found that if a randomized search is repeated many times, that a small number of variable settings were shared by all solutions. They also found that if they set those variables before conducting the rest of the search, then formerly exponentially runtimes collapsed to low-order polynomial time. They called these shared variables the *backdoor* to reducing computational complexity.

Combining the above, we propose the following strategy for faster reasoning about RE models:

1) Use random search with retries to find the backdoors (which we call the "key" decisions) in RE models;
2) Debate, then decide, about the keys before exploring anything else;
3) To avoid trivially small solutions, our random search should strive to cover much of the model.
4) Accordingly, we implement this strategy within a multi-objective optimizer that seeks to maximize goal coverage, while at the same time minimizing the number of conflicts and minimizing the sum of the costs of the decisions made within that model.

---

1. When considering an edge E_j from N_i to a child N_k then:
- If $N_i \in \{1, -1\}$ then expect $N_k = E_j * N_i$ else
- If $E_j > 0$ then expect $N_k = N_i$ else
- If $E_j < 0$ then expect $N_k = -N_i$.

Goal model inference does not use a fuzzy set or probabilistic approach to reasoning about conflicting influences. Rather, goal edges are either used or *ignored*. Hence:

2. If N_k is *undefined*, then it is labelled with the above expectations and we call STEP recursively over all edges to N_k's children. Children are explored in a random order. After recursion, for and-nodes, labels get set back to *undefined* if any fail N_k expectations (as defined by point #1).

3. Otherwise, if its label does not meet the above expectations, then we *ignore* the edge E_j.

Fig. 5. Procedure STEP: labels neighboring nodes; may *ignore* some edges. Called by the SAMPLE procedure of Fig. 6.

V. Goal Inference with SHORT

In order to test the value of the strategy mentioned above, we need case studies. The largest RE models we could access were goal models generated by Jennifer Horkorff and Steve Easterbrook (see goo.gl/K7N6PE) and are listed in Table I. All these models have the structure shown in Fig. 3. For images of the models, see goo.gl/K7N6PE.

Fig. 4 gives a high-level overview of how these models are explored. Our SHORT tool runs in five phases:

SH. Sample Heuristically the possible labellings.

O. Optimize the label assignments in order to cover more goals or reduce the sum of the cost of the decisions in the model. To implement the *retry* step recommended recommended by Crawford and Baker's ISAMP [12].

R. Rank all decisions according to how well they performed during the optimization process.

T. Test how much conclusions are determined by the decisions that occur very early in that ranking.

SH = Sample Heuristically: The SAMPLE procedure of Fig. 6 finds one possible set of consistent labels within a goal. The procedure calls the STEP procedure (Fig. 5) over all the decisions in the model, labeling as many nodes as

1. SAMPLE inputs (a) a goal model and (b) a set of *prior* decisions made by any previous run of SAMPLE (so initially, this set is empty).
2. As it initializes, SAMPLE sets all nodes to *undefined* then all *prior* nodes to *satisfied*;
3. While there are *undefined* nodes, SAMPLE (1) "picks" one *undefined* decision as *satisfied* then (2) "reflects" over its edges using the STEP procedure of Fig. 5. SAMPLE is stochastic since "picks" and "reflects" return nodes and edges in a random order.
4. SAMPLE outputs a *solution* listing all *satisfied nodes*.

Important note: if *prior* is changed, then SAMPLE will return different solutions. That is, the results from SAMPLE can be carefully tuned and improved by the OPTIMIZE procedure of Fig. 7 that carefully selects useful *prior*s.

Fig. 6. Procedure SAMPLE: tries to labels many nodes. Called by the OPTIMIZE procedure of Fig. 7.

possible. This procedure executes in a random order since (a) this emulates the idiosyncrasies of human discussions; (b) as mentioned above, Crawford and Baker found this to be a useful strategy for reducing computational complexity.

Sometimes we are asked why use SAMPLE when there are some many other soft goal inference methods in the literature; e.g. the forwards and backwards analysis proposed by Horkoff and Yu [9] (see *Related Work*, earlier in this paper). One reason we prefer SAMPLE is its generality. SAMPLE was first developed for Menzies' HT4 system and has been applied to all the model types listed above under "Question 2" of §II; i.e. a) causal diagrams; (b) qualitative equations; a (c) frame-based knowledge representation; (d) compartmental models, and even (e) first-order systems (with finite-domains on the variables) [16].

O = Optimize: Fig. 6 discusses how we SAMPLE one set of labels from a softgoal model. As described at the bottom of that figure, that SAMPLE-ing process can be controlled via the *prior* decisions passed to the model. The OPTIMIZE procedure of Fig. 7 is a multi-objective evolutionary algorithm that learns what *prior*s decisions lead to *better* labellings. Here, *better* means labels that result in (a) greater coverage of goals and softgoals, (b) minimization of skipped edges; (c) least cost solutions. Note: for this paper, we assume that all decision costs are equal (in future work, we will explore other distributions).

R= Rank: The RANK procedure of Fig. 8 reflects of the results of the optimizer to rank each decisions. Decisions are ranked by the odds that they are associated with the better goals. Note that the key decisions will the ones found in the first part of those decisions.

T = Test: The TEST procedure of Fig. 9 takes a ranked list of decisions, then tests what happens when the first few are fixed and the rest are selected at random.

A. Reporting the Results

The SHORT process described above is a large-scale "what-if"+optimization procedure. In order to succinctly describe the results of the above, we apply the following statistical summarization method.

1. Given a model with n decisions, OPTIMIZE calls SAMPLE $N = 10 * n$ times. Each call generates one member of the population $pop_{i \in N}$.
2. OPTIMIZE scores each pop_i according to various objective scores o. In the case of our goal models, the objectives are o_1 the sum of the cost of its decisions, o_2 the number of ignore edges, and the number of o_3 satisfied goals and o_4 softgoals.
3. OPTIMIZE tries to each replace pop_i with a mutant q built by extrapolating between three other members of population a, b, c. At probability p_1, for each decision $a_k \in a$, then $m_k = a_k \vee (p_1 < rand() \wedge (b_k \vee c_k))$.
4. Each mutant m is assessed by calling SAMPLE(*model,prior=m*); i.e. by seeing what can be achieved within a goal after first assuming that $prior = m$.
5. To test if the mutant m is preferred to pop_i, OPTIMIZE uses Zitler's continuous domination *cdom* predicate [33]. This predicate compares two sets of objectives from sets x and y. In that comparison, x is better than another y if x "losses" least. In the following, "n''" is the number of objectives and $w_j \in \{-1, 1\}$ shows if we seek to maximize o_j.

$$
\begin{aligned}
x \succ y &= loss(y, x) > loss(x, y) \\
loss(x, y) &= \sum_{j}^{n} -e^{\Delta(j,x,y,n)}/n \\
\Delta(j, x, y, n) &= w_j(o_{j,x} - o_{j,y})/n
\end{aligned}
$$

5. OPTIMIZE repeatedly loops over the population, trying to replace items with mutants, until new better mutants stop being found.
6. Return the population.

Fig. 7. Procedure OPTIMIZE: strives to find "good" priors which, when passes to SAMPLE, maximize the number of edges used while also minimizing cost, and maximizing satisfied hard goals and soft goals. OPTIMIZE is based on Storn's differential evolution optimizer [34]. OPTIMIZE is called by the RANK procedure of Fig. 8. For the reader unfamiliar with the mutation technique of step 3 and the *cdom* scoring of step 5, we note that these are standard practice in the search-based SE community [35, 36].

1. Run OPTIMIZE $N = 20$ times, keep all decisions d in the generated population $pop_{i \in N}$ and their objectives o.
2. For each objective $o_j \in o$, do
   - Sort and separate the top 10% o_j scores; call that "best" b and the remainder "rest".
   - For each decision $d_k \in d$, do
     - Count how many times n_1 that d_k is associated with a "best" objective score. Set $n_2 = N - n_1$
     - The value of d_k for objective o_j is the probability p times the support s that d_k appears more often in "best" than "rest". That is, $s_{j,k} = n_1 * 0.1$ and $p_{j,k} = (n_1 * 0.1)/(n_1 * 0.1 + n_2 * 0.9)$.
3. Let *ordering* be all decisions, sorted descending by the value of decision d_k across all objectives; i.e. $v_k = \sum_j s_{j,k} \times p_{j,k}$

Fig. 8. Procedure RANK: ranks all decisions according to how well they performed during the optimization process. Used by the TEST procedure of Fig. 9.

Fig. 10 shows what is reported by TEST after studying the modernization model of Fig. 2. At each point along the x-axis, TEST samples the goal model 20 times using *prior* decisions taken from $1 \leq i \leq x$. In the solutions returned by SAMPLE, *root cost* is the sum of decision costs; *root goal/softgoals* are the sum of the number of satisfied goals/soft goals. The blue and red plots show the

1. Run RANK to get a sorted list of decisions d. 2. For all
decisions $1 \le x \le |d|$, do

- Build a *prior* set using decisions $d_1..d_x$ from *ordering*;
- 20 times, call *SAMPLE(model, prior)*
- Record for position x the median and IQR of the
  objectives found in those results.

Aside: median= 50th percentile; IQR=(75th-25th) percentile.

Fig. 9. Procedure TEST: from a ranked list of decisions d, set the
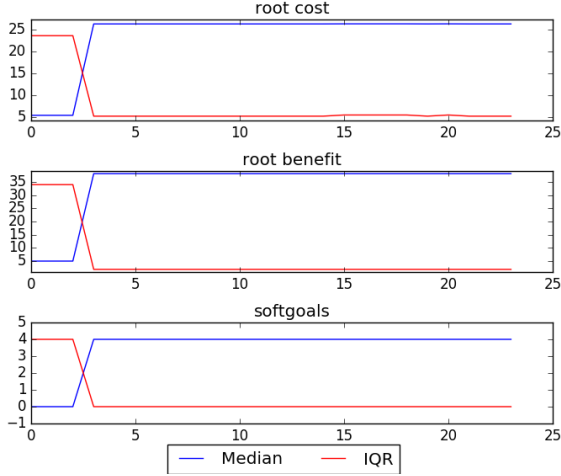first few decisions, then selected the rest at random.

Fig. 10. The X-axis contains d decisions sorted by the RANK
procedure of Fig. 8. The y-axis shows the results from the TEST
procedure of Fig. 9; i.e. for $1 \le x \le |d|$, fix the first x decisions then,
20 times, make random choices about decisions $x + 1$ to $|d|$. Results
shown as median and IQR values. Median = 50% percentile and
IQR=intra-quartile range= (75-25)th percentile. These results are
smoothed such that these plots only change where this is statistically
significant non-small change in the y-axis. An appendix to this paper
describes our statistical smoothing procedure.

median and variation around the median.

In these results, after making three decisions, the varia-
tions plummet and the medians rise to a steady plateau.
Fig. 10 is reporting that this model has three key decisions
which, if set, make further discussion superfluous. Note
that these results match what we learned from a visual
inspection of Fig. 2 in §II.

The decision rankings found by RANK for Fig. 2 are
shown in Fig. 11. Two features of this list deserve comment:

- This list is *not* 24 decisions that could inspire $2^{24} >$
  $16,000,000$ debates. Rather, it is an *ordering* with
  the property that item $x + 1$ is recommended *only* if
  recommendations $1..x$ are first adopted. That is, this
  list offers only 24 decisions to users (whether or not
  they which to perform actions $1..x$).
- If users cannot implement all these recommendations,
  they can easily read from Fig. 10 the effects of
  implementing just the first x items.

1) J2EE Specification (satisfied)
2) *Pnp Framework (denied)*
3) *New Database (denied)*
4) Documentation Tool (satisfied)
5) Access Control Assessed (satisfied)
6) Monitoring Pilot (satisfied)
7) General Test Env (satisfied)
8) Bakeoff Result (satisfied)
9) Access Control Pilot (satisfied)
10) DB Vendor Test Env (satisfied)
11) Data Service Spec (satisfied)
12) External clients get their request (satisfied)
13) Co-ordinates & internal client (satisfied)
14) Co-ordinates & external client (satisfied)
15) Data Model Pilot (satisfied)
16) Data Service Pilot (satisfied)
17) 2 Tier (satisfied)
18) 3 Tier (satisfied)
19) Define data model for shared data (satisfied)
20) *Svc layer w/ extracted biz logic (denied)*
21) Define ext mandatory data std (satisfied)
22) Svc layer w/ extracted biz logic in DB (satisfied)
23) External data model can be extended (satisfied)
24) Provide logical data scheme internally (satisfied)

Fig. 11. Results of decision ordering on Fig. 2. Read "satisfied/denied"
as equivalent to ensuring that a leaf node is achieved or not achieved.
Note the *denied* items at positions $x \in \{2, 3, 20\}$. These are recommen-
dations of what *not* to do. The rest are all positive recommendations.
The implication of Fig. 10 is that after following the recommendations
for the first 3 decisions, the remaining decisions will have little
additional impact on overall cost or satisfaction.

Model	Result	Percentage of Decisions					
		0	6	12	25	50	100
Counselling	SGs	60	60	70	80	70	60
	Goals	50	60	70	60	70	40
	Costs	n.a.	5	10	20	28	35
Management	SGs	50	50	60	60	50	40
	Goals	50	50	50	60	60	60
	Costs	n.a.	4	8	20	24	32
Marketing	SGs	70	70	70	60	60	30
	Goals	70	70	70	70	60	40
	Costs	n.a.	6	8	11	12	20
ITDepartment	SGs	70	70	60	70	70	60
	Goals	60	50	50	80	80	70
	Costs	n.a.	2	4	9	17	18
SAProgram	SGs	80	80	80	80	80	80
	Goals	70	70	70	60	55	60
	Costs	n.a.	1	2	3	3	5
Services	SGs	50	50	50	50	50	50
	Goals	30	50	60	60	70	70
	Costs	n.a.	4	7	13	19	24
Kids&Youth	SGs	50	40	50	50	40	20
	Goals	50	50	70	60	60	70
	Costs	n.a.	0	0	4	7	10

TABLE II

PERCENTAGES OF SOFTGOALS AND GOALS COVERED, AND TOTAL
COSTS, WITH RESPECT TO THEIR MAXIMUM VALUES AFTER APPLYING
TOP 6,12,25,50,100% OF DECISIONS FOUND AND RANKED BY SHORT
FOR THE GOAL MODELS OF TABLE I.

VI. LOOKING FOR KEYS IN RE MODELS

In this section, we check for the presence of a small
number of key decisions in numerous other RE models.

The Fig. 2 model has 53 nodes and 57 edges. Table I shows
some details on the other goal models used in this study.
The largest of our sample is the *CSServices* models shown in
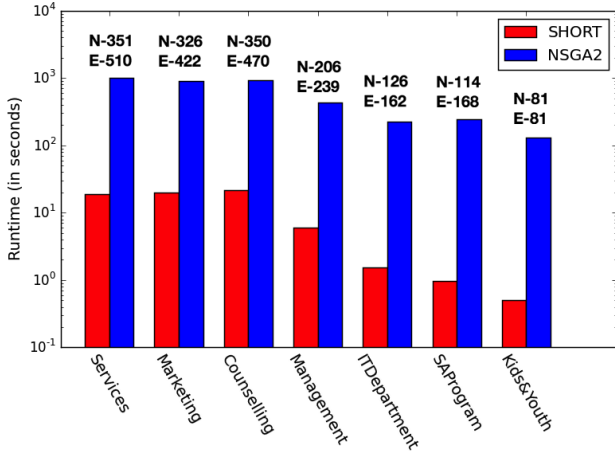Fig. 1. For images of all these models, see goo.gl/K7N6PE.

Fig. 12. Runtimes for nsga2 compared to our approach. The number of nodes(N) and the edges(E) for each model is shown above the respective bar.

Table II shows SHORT's results after making the first 6,12,25,50,100% of the decisions along the ranking found by RANK. In a result consistent with Fig. 10, there is little change to goal or soft goal coverage are making just a few decisions: typically, just 12% of the decisions.

Two exception to the pattern "12% is enough" are CSCounselling and ITDepartment. Those models achieved nearly max goal coverage at 12% but did not peak until making 25% of the decisions. Even with that exception, the general conclusion is clear: with SHORT, only a minority of the decisions need to be made with care, since once those are made, the goal coverage is robust (unchanging) for the remaining decisions.

(Aside: one quirk of Table II is that, as shown in the column headed with "0", t even when no decisions are made, it is possible to achieve some of the goals and softgoals, just by making decisions at random. Since no human has committed to any decision in that region, we mark all the 0% costs as "n.a.".)

Fig. 12 shows the runtimes required to generate our results. Empirically, these runtimes fit the curve $secs = nodes^2$) with an $R^2 = 0.97$; i.e. SHORT's runtimes are low-order polynomial. This is a significant result since in 1990, Bylander et al. warned that abductive search is NP-hard [31]; i.e. when exploring all options within an RE model, we should expect very slow runtimes. The pessimistic result has confirmed empirically by Menzies *et al.* [37], theoretically by Abdelbar *et al.* [32], and then again empirically by Ernst *et al.* [15].

It is insightful to compare SHORT's runtimes against the *forward* and *backwards* analysis proposed by Horkoff and Yu [9]. They report times in the range of 7 to 300 seconds (which includes human time considering various choices)—which is approximately the same as the runtimes seen in Fig. 12. The advantage of backward and forward analysis is that user involvement increases user acceptance

of the conclusions. The disadvantage is that those analysis methods do not comment on the robustness of the solution. Further, a forward and backwards analysis results in one sample of the model. A SHORT-style analysis, on the other hand, includes extensive "what-if" simulations. Charts like Fig. 10 not only offer solutions but also comment on:

- The stability of the selected decisions: see the red "IQR" line in that figure;
- The trade space across multiple decisions. Users can check a display of SHORT's decision orderings (like Fig. 10) to decide for themselves when enough benefit has been obtained for enough cost.

To the best of our knowledge, forward and backward analysis has not been benchmarked against alternate techniques. Our next results compare SHORT against a state-of-the-art multi-objective optimizer called NSGA-II [38]. This is a widely used genetic algorithm that uses a novel *select* operator to find the best "parents" to make the next generation. For both NSGA-II and SHORT, the goal is to maximize the coverage of goals and soft goals, while at the same time minimizing the sum of the costs of making the decisions. Fig. 12 shows the median runtime for *one* run of NSGA-II and SHORT. Note that SHORT runs 2-3 orders of magnitude times faster.

In addition, NSGA-II returns just one solution, whereas SHORT reports what happens when an increasing number of decisions are imposed on a system (as done in Fig. 10). In order for NSGA-II to reason like SHORT, it would have to run hundreds of 'what-if' studies. That is, NSGA-II's runtime costs would be incurred hundreds of times.

To explain SHORT's faster runtimes, we invoke the same reasoning used by Crawford and Baker in §IV-A. The RE models contain a small set of key variables that set the rest. Hence, rigorously searching through all variable settings is not recommended since only a small number of those settings actually matter. Further, when the key decisions variables are spread thinly over the entire model, it makes no sense to carefully explore many parts of the model (as done by NSGA-II) since much time will be wasted "walking" between the far-flung keys. For such models, if the reasoning gets stuck in one region, then the best thing to do is to leap at random to some distant part of the model (as done by SHORT's random search).

Table III shows what objectives were reached in 20 repeated runs of NSGA-II and SHORT. Both approaches achieved remarkably similar coverage of goals—an effect that can be explained by our models having a small number of key variables which, if set the right way, control what can be achieved in the rest of the model.

VII. Threats to Validity

As with any empirical study, biases can affect the final results. Therefore, any conclusions made from this work must be considered with the following issues in mind:

Sampling bias threatens any experiment; i.e., what matters in (say) practitioner settings may not be true of our

Model	NSGA2	SHORT
Services	f1: 46.77 ± 0.0 f2: 65.22 ± 0.0	f1: 46.24 ± 0.0 f2: 65.22 ± 0.0
Marketing	f1: 32.47 ± 0.0 f2: 34.38 ± 0.0	f1: 32.47 ± 0.0 f2: 34.38 ± 0.0
Counselling	f1: 54.29 ± 1.41 f2: 44.83 ± 3.45	f1: 53.59 ± 2.76 f2: 44.83 ± 3.45
Management	f1: 44.63 ± 0.82 f2: 61.11 ± 2.78	f1: 42.98 ± 1.65 f2: 61.11 ± 2.78
ITDepartment	f1: 68.42 ± 2.63 f2: 69.57 ± 0.0	f1: 68.42 ± 2.63 f2: 73.91 ± 8.69
SAProgram	f1: 78.69 ± 1.64 f2: 66.67 ± 0.0	f1: 78.69 ± 1.64 f2: 66.67 ± 0.0
Kids&Youth	f1: 22.86 ± 0.0 f2: 100.0 ± 0.0	f1: 22.86 ± 0.0 f2: 100.0 ± 0.0

TABLE III
NSGA-II COMPARED TO OUR SHORT APPROACH. COLUMNS REPORT ACCURACY(MEDIAN ± IQR) OF THE OBJECTIVES *f1*: PERCENTAGE OF SOFT GOALS SATISFIED AND *f2*: PERCENTAGE OF GOALS SATISFIED.

examples. The data sets used here comes from goal models taken from a repository and all but one were supplied by one individual. These were all from i* style models. That said, as argued in §II, the models we use here share many aspects with RE representations used by many other researchers.

Evaluation bias: This paper uses one measure of success, goal achievement and associated cost and benefit. Optimal solutions only consider inputs given, and may not reflect all the complexities of a given decision.

Construct Validity: The conclusions of this paper are about better processes for making *decisions* (specifically, do not waste time on all the numerous redundant issues). It would be a violation of construct validity to make another claim—that SHORT-guided decisions lead to better outcomes.

VIII. CONCLUSIONS

Confusing models can confuse stakeholders. One way to untangle complex models is to find the small number of key decisions that determine what can be done for the rest of the model.

Such keys are naturally occurring in RE models. For example, in eight large RE models sampled from the the i* community, we have found a small number of decisions (often, just 12%) was enough to to set the rest of the decisions. In stark contrast to much prior work [14, 15], finding these keys was very fast, using the randomized search methods employed in SHORT. Using these keys, SHORT is able to optimize RE models in time that was orders or magnitude faster than state-of-the-art optimizers. Further, the optimizations found by SHORT were competitive with those found by other, much slower, methods. Hence, our conclusions are that:

1) Keys can be found in numerous RE models;
2) Keys are easy to find;
3) It is very useful to apply keys for reasoning about RE models.

TECHNICAL APPENDIX

Reproduction Package

All our tools and models are available on-line at https://goo.gl/gvxeaH. While some aspects of our approach are specific to goal models, the general SHORT method could be applied to a wide range of models.

Graph Smoothing

To smooth out the charts generated in SHORT (e.g. Figure 10), we use the Scott-Knott procedure recommended by [39]. This technique recursively bi-clusters a sorted set of numbers. If any two clusters are statistically indistinguishable, Scott-Knott reports them both as one line.

Scott-Knott first looks for a break in the sequence that maximizes the expected values in the difference in the means before and after the break. More specifically, it splits l values into sub-lists m, n in order to maximize the expected value of differences in the observed performances before and after divisions. E.g. for lists l, m, n of size ls, ms, ns where $l = m \cup n$, Scott-Knott divides the sequence at the break that maximizes:

$$E(\Delta) = \frac{ms}{ls} abs(m.\mu - l.\mu)^2 + \frac{ns}{ls} abs(n.\mu - l.\mu)^2$$

Scott-Knott then applies some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each division. For this study, our hypothesis test H was a conjunction of the A12 effect size test (endorsed by [40]) and non-parametric bootstrap sampling [41]; i.e. our Scott-Knott divided the data if *both* bootstrapping and an effect size test agreed that the division was statistically significant (99% confidence) and not a "small" effect ($A12 \geq 0.6$).

REFERENCES

[1] L. Chung, J. Mylopoulos, and B. A. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach," *IEEE Transactions on Software Engineering*, vol. 18, pp. 483–497, 1992.
[2] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, Toronto, 2001, pp. 249–263.
[3] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. S. K. Yu, "Evaluating Goal Models within the Goal-oriented Requirement Language," *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 841–877, 2010.

[4] H. Kaiya, H. Horai, and M. Saeki, "AGORA: Attributed Goal-Oriented Requirements Analysis Method," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, Essen, Germany, 2002, pp. 13–22.

[5] I. J. Jureta, A. Borgida, N. Ernst, and J. Mylopoulos, "Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, Sydney, Australia, 2010, pp. 115–124.

[6] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, Annapolis, Maryland, 1997, pp. 226–235.

[7] H. A. Simon, *The Sciences of the Artificial*. MIT Press, 1996.

[8] T. P. Moran and J. M. J. M. Carroll, *Design rationale : concepts, techniques, and use*, ser. Computers, cognition, and work. Mahwah, N.J. (Etats-Unis): L. Erlbaum, 1996.

[9] J. Horkoff and E. Yu, "Interactive goal model analysis for early requirements engineering," *Requirements Engineering*, vol. 21, no. 1, pp. 29–61, 2016.

[10] R. Williams, C. P. Gomes, and B. Selman, "Backdoors to typical case complexity," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[11] S. Amarel, "Program synthesis as a theory formation task: problem representations and solution methods," in *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, 1986.

[12] J. M. Crawford and A. B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, ser. AAAI'94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 1092–1097.

[13] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, no. 1-2, pp. 273–324, Dec. 1997.

[14] T. Menzies, "Applications of abduction: knowledge-level modelling," *International journal of human-computer studies*, vol. 45, no. 3, pp. 305–335, 1996.

[15] N. Ernst, A. Borgida, J. Mylopoulos, and I. J. Jureta, "Agile Requirements Evolution via Paraconsistent Reasoning," in *International Conference on Advanced Informations Systems Engineering*, Jun. 2012, pp. 1–16.

[16] T. Menzies, "Applications of abduction: knowledge-level modelling," *International Journal of Human-Computer Studies*, pp. 305–335, 1996.

[17] Y. Wang and J. Mylopoulos, "Self-repair through reconfiguration: A requirements engineering approach," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2009, pp. 257–268.

[18] A. van Lamsweerde, "Requirements engineering: from craft to discipline," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 238–249.

[19] A. Classen, P. Heymans, and P.-Y. Schobbens, "WhatâĂŹs in a feature: A requirements engineering perspective," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2008, pp. 16–30.

[20] M. Mendonca, A. Wąsowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 231–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753235.1753267

[21] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés, "Fama: Tooling a framework for the automated analysis of feature models." *VaMoS*, vol. 2007, p. 01, 2007.

[22] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and Minimum-Cost Satisfiability for Goal Models," in *Proceedings of the International Conference on Advanced Information Systems Engineering*, Riga, Latvia, 2004, pp. 20–35.

[23] J. Horkoff and E. Yu, "Comparison and evaluation of goal-oriented satisfaction analysis techniques," *Requirements Engineering*, vol. 18, no. 3, pp. 199–222, 2013.

[24] C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Multi-objective reasoning with constrained goal models," *Re-quirements Engineering*, pp. 1–37, 2016.

[25] T. Menzies, D. Owen, and J. Richardson, "The strangest thing about software," *IEEE Computer*, pp. 54–60, 2007.

[26] T. Menzies and H. Singh, "Many maybes mean (mostly) the same thing," in *Soft Computing in Software Engineering*, ser. Studies in Fuzziness and Soft Computing. Springer, 2004, vol. 159, pp. 125–150.

[27] M. Druzdzel, "Some properties of joint probability distributions," in *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence*, 1994, pp. 187–194.

[28] B. Nuseibeh, "To be and not to be: On managing inconsistency in software development," in *Proceedings of the 8th International Workshop on Software Specification and Design*. IEEE Computer Society, 1996, p. 164.

[29] D. Poole, "Who chooses the assumptions?" in *The Management of Uncertainty in AI, IJCAI'93*, 1993.

[30] J. De Kleer, "An assumption-based tms," *Artificial intelligence*, vol. 28, no. 2, pp. 127–162, 1986.

[31] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson, "The computational complexity of abduction," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 25–60, 1991.

[32] A. M. Abdelbar, "Approximating cost-based abduction is np-hard," *Artif. Intell.*, vol. 159, no. 1-2, pp. 231–239, Nov. 2004.

[33] E. Zitzler and S. Künzli, *Indicator-Based Selection in Multiobjective Search*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 832–842.

[34] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[35] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[36] J. Krall, T. Menzies, and M. Davies, "Gale: Geometric active learning for search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, pp. 1001–1018, 2015.

[37] T. Menzies, "Applications of abduction: knowledge-level modelling," *International Journal of Human-Computer Studies*, vol. 45, no. 3, pp. 305 – 335, 1996.

[38] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Trans. Evol. Comp*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[39] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 537–551, 2013.

[40] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the ACM/IEEE International Conference on Software Engineering*. IEEE, 2011, pp. 1–10.

[41] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*, ser. Mono. Stat. Appl. Probab. London: Chapman and Hall, 1994.