An Integrated Approach to a Combinatorial Optimisation Problem*

J. Bowles[0000−0002−5918−9114] and M. B. Caminati[0000−0002−4529−5442]

School of Computer Science, University of St Andrews
St Andrews KY16 9SX, United Kingdom
{jkfb|mbc8}@st-andrews.ac.uk

Abstract. We take inspiration from a problem from the healthcare domain, where patients with several chronic conditions follow different guidelines designed for the individual conditions, and where the aim is to find the best treatment plan for a patient that avoids adverse drug reactions, respects patient's preferences and prioritises drug efficacy. Each chronic condition guideline can be abstractly described by a directed graph, where each node indicates a treatment step (e.g., a choice in medications or resources) and has a certain duration. The search for the best treatment path is seen as a combinatorial optimisation problem and we show how to select a path across the graphs constrained by a notion of resource compatibility. This notion takes into account interactions between any finite number of resources, and makes it possible to express non-monotonic interactions. Our formalisation also introduces a discrete temporal metric, so as to consider only simultaneous nodes in the optimisation process. We express the formal problem as an SMT problem and provide a correctness proof of the SMT code by exploiting the interplay between SMT solvers and the proof assistant Isabelle/HOL. The problem we consider combines aspects of optimal graph execution and resource allocation, showing how an SMT solver can be an alternative to other approaches which are well-researched in the corresponding domains.

1 Introduction

In complex systems it is common for processes to execute in parallel. The underlying composed behavioural model is complex and, unless strictly necessary, it is preferable to avoid computing it. We developed an approach to find optimal paths across multiple models denoting preferred scenarios of execution. The choice of a path across a model may be influenced by external factors, available resources and further constraints on such resources. Models are given as directed graphs, and the challenge is to find paths across all graphs satisfying not just all constraints but optimised against one or more arithmetic measures.

* This research is supported by MRC grant MR/S003819/1 and Health Data Research UK, an initiative funded by UK Research and Innovation, Department of Health and Social Care (England) and the devolved administrations, and leading medical research charities.

Our approach has been inspired by a problem in healthcare where patients with two or more ongoing chronic conditions, also known as *multimorbidity*, do not receive adequate treatment. Typically patients follow treatment guidelines for individual conditions in isolation, and the presence of multimorbidity requires them to follow several guidelines simultaneously. However, there is a lack of guidance on how best to prioritise recommendations for multimorbidity [17]. Patients with multimorbidity are often required to take many medications for their conditions, which can cause adverse drug reactions often leading to unnecessary complications and hospitalisations [15]. It is also possible that taking several medications together decreases the effectiveness of the individual drugs when administered at the same time, or that a patient is intolerant to a combination of medications. In precision therapeutics, the aim is to tailor medical treatment to the individual characteristics of each patient which includes finding the right set of medications for patients with multimorbidities.

Besides the small example in Section 2, where the healthcare setting is used for illustrative purposes, we abstract from the medical problem and terminology in this paper, and it suffices to understand each guideline as a directed graph whereby a patient may be at a given node at a particular moment in time, resources are (groups of) medications (aka drugs), and external factors can denote patient specific intolerances, and so on.

Resources have a known value of how effective they are individually for the purpose they are used for (e.g., metformin when used for treating diabetes), as well as a measure of conflict (interaction constraints) with others. Our aim is to find the optimal path across all graphs which maximises effectiveness and minimises conflicts. Such a path selects the drugs across multiple graphs that suit the patient best.

This paper is structured as follows. Section 2 motivates and describes the context of our approach in more detail. Related work is described in Section 3. Section 4 describes the abstraction used for capturing guidelines as directed graphs, and how paths and interaction constraints are defined. Our translation to SMT (Satisfiability Modulo Theories) code is shown in Section 5 and how verification is done is described in Section 6. This is followed by concluding remarks in Section 7.

2 Problem motivation and informal description

Assume that a patient with an acute condition is hospitalised on day 0. There are two possible treatments for the condition: a non-surgical treatment and surgery. The two alternatives are represented by the two branches in the directed graph of Fig. 1 (left), where the source node represents the hospitalisation. The right branch represents the choice of surgery with nodes n_3 and n_4 denoting the steps implied by this choice (n_3: pre-surgical testing and n_4: the surgery itself). Each node execution in a treatment graph may correspond to a choice of actions. For example, in the case of pre-surgical testing it involves administering one of two drugs (d_1 or d_2), while in the case of the surgery, only one possibility is

present. The left branch (with n_2) models the non-surgical choice, here associated to the prescription of drug d_0 (with no other choice available). The bracketed number besides each node represent its duration: for example, pre-surgical testing (node n_3) lasts 2 days. Furthermore, this particular patient suffers from two

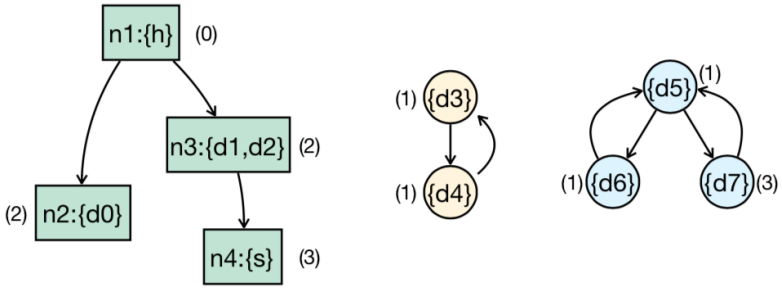

Fig. 1. A simple example problem

chronic conditions: C_1 requires him to take drug d_3 on even days and d_4 on odd days (Fig. 1 middle), and C_2 requires him to take drug d_5 on even days followed by either d_6 or d_7 on odd days whereby d_7 further delays the subsequent administering of d_5 by two days (Fig. 1 right).

Surgery is preferable, but d_1 interacts negatively with d_3 and d_2 with d_4. It is known that d_6 is more effective than d_7 when used for treating C_2. However, d_7 can act as an inhibitor in the reaction between d_1 and d_3, mitigating the ensuing negative effects on the patient. In this fictitious example, we want to determine the best course of action in the hospitalisation of a patient with chronic conditions C_1 and C_2 taking into account drug efficacy and known drug interactions.

The simple example above expresses the family of problems we want to deal with. In what follows, we abstract away from the healthcare application domain expressed above in terms of directed graphs, one for each separate treatment a patient undergoes, and where each node is annotated with a *duration* and a potential choice of *resources*. This makes it possible to reduce the problem to a path finding problem across several directed graphs, respecting constraints on interactions among several resources and their temporal overlapping. In other words, we want to find paths in given directed graphs (*digraphs*, [2,9]) such that:

P1 for each visited node a resource (e.g., in our context this can be a medication) is selected amongst a set of allowed ones for that node, and
P2 the interactions between selected resources are minimised and their individual effectiveness maximised in the obtained solution.

The effect of the mutual interactions (see point (P2) above) on the computation of the solution takes into account the duration of each node which, in turn,

determines the required temporal distance between possibly interacting resources (e.g., two interacting resources happening at close temporal distance can be given more weight than the same interacting resources happening at distant times). The efficiency of modern SMT solvers makes them feasible to search for a solution satisfying the criteria above. However, only (P2) above makes use of the arithmetic capabilities of the solver, while we otherwise only made use of the basic SAT solver for the remaining computations in our previous work [5,6,7]. Given that SMT solvers are SAT solvers with awareness of additional facts (e.g., arithmetics), we wonder whether this can be used to better exploit SMT solvers capabilities. In our case, motivations for working in this direction include:

M1 Given that resource interactions are expressed by numerical functions having the resources as arguments, the current basic SAT implementation only allows us to consider binary interactions as opposed to interactions of an arbitrary number of resources.

The limitation to binary interactions arises because the arity of SMT functions is fixed, and more elaborated datatypes such as lists or function operators are not directly available. We will see in Section 5 how this limitation can be worked around by the theory of Bit-Vectors through which the SMT-LIB standard [3] extends the basic domain of propositional logic in which a standard SAT solver operates. We use this theory to emulate a variadic function, i.e., a function taking a variable number of variables, thereby granting the possibility of expressing interactions between a variable number of resources. This makes it possible to express the particular case of non-monotonic interactions whereby adding a resource to, say, a pair of conflicting resources not necessarily exacerbates the conflict. The interest in this was illustrated in our healthcare example earlier, where drug d_7 inhibits an interaction between two other drugs.

M2 Expressing the problem of path finding in SAT terms leads to assertions rapidly increasing in complexity with the dimension of the given graphs, and hence unintelligible SAT code.

We addressed the problem through a novel combination of the SMT solver itself and the theorem prover Isabelle [21], formally proving the correctness of the complicated SAT code used for path finding. However, any path naturally admits a discrete linear ordering, which suggests an alternative route to expressing the path finding problem in the SMT-LIB language: the solution, rather than consisting of a function over the nodes being true exactly for all the nodes in the path, can be expressed by a function from the integers into the set of nodes, telling, when its argument has value i, which node is the i-th in the path. This is possible because SMT solvers, contrary to SAT solvers, are aware of the theory of integers (i.e., in the latter, arithmetic operators can only be introduced as uninterpreted operators). For this strategy to work, one must be able to express the requirement that the $i+1$-th node visited in the path belongs to the set of children of the i-th node visited. However, the notion of "set" and "belongs to" are not directly available in SMT solvers: we will see in Section 5 how, again, the

theory of Bit-Vectors allows us to find a workaround to this problem which, in our previous work, caused increased complexity in the produced SMT code.

Furthermore, we note that sorting path nodes by exploiting the natural number linear ordering allows us to restrict the scope of the sought solution. This is useful since in practice we are often only interested in considering a solution for future steps not too distant from the current one.

3 Related work and justification of an SMT approach

The problem we propose features (as we will more formally express in Section 4) path finding across multiple directed (non necessarily acyclic) graphs, resources selection among sets of resources annotating each path-belonging node, overall optimisation of the selected resources under constraints of arbitrary arity, and a dependency of these constraints on the temporal separation of selected resources.

The fact that it combines traits of the distinct, widely explored problems in *combinatorial optimisation* (CO) which are mentioned above does not imply that the corresponding, well-explored solutions for these distinct problems can be easily combined to obtain a solution to the problem considered in this paper. For example, one could try to apply existing, well-known algorithms for the path-finding portion of our problem, and then hope to reduce the remaining portions of the problem to well-understood problems in CO, maybe by applying integer programming (IP), one of the domains of CO richest in methods and algorithms, to the sub-problem of resource subset selection. However, the problem of resources subset selection is not natively expressed in terms of integer variables and functions; and, more importantly, even if one finds a IP rendition of it, that rendition is bound to be, in general, non-linear, due to the fact that our notion of resource interactions aims by the design at dealing with the non-monotonic case (in more formal terms, the function ι we will introduce in (4) is completely arbitrary). This implies that we do not have direct access to the powerful and well-developed methods of integer linear programming (ILP), and have to deal with the more difficult approaches available in the less developed field of non-linear programming, where typically one has to resort to techniques (reconnecting the problem to special cases) such as continuous relaxation, branch and bound algorithms, approximation algorithms, ad-hoc heuristics, fractional programming [16,8,1]. All these approaches typically require significant amounts of work, add complexity to the solutions and require assumptions about the problem specification (for example, concavity/convexity requirements on the continuous relaxation's feasible region).

Before choosing the problem treatment presented in this paper, as explained above in the case of IP, we reviewed many other combinatorial optimisation sub-domains, without finding techniques able to capture all the aspects of the problem we consider. For example, resource scheduling [22, Section 22], an actively researched field, deals with executing a set of activities, each needing to employ some resources, while respecting the resource capacities, temporal constraints, and while optimising a given objective function. While this can accommodate

the concepts of nodes (as activities) and resources featured in our problem, the focus about the constraints in resource scheduling is on the capacities of the resources and maybe (through the objective function) on temporal optimisations (e.g., minimising the number of late activities); in contrast, in our problem, the optimisation focuses on the mutual (and non necessarily binary) interaction of resources, as we will see in Section 4 (e.g. formula (8) featured there).

The considerations above about the difficulties we encountered in adapting existing techniques in the broad domain of combinatorial optimisation to the several facets of the problem at hand suggests that this problem presents some degree of hardness. This is confirmed by noting that the simpler problem of finding a path in a single graph with the additional constraint of the binary exclusion (e.g., on an either selected or unselected basis, as opposed to our problem featuring a score-based selection criterion) of given sets of node pairs is established to be NP-hard in [14].

Further justification for the adoption of SMT techniques is given by the fact that other authors also employ SAT or SMT solvers for simpler problems. SMT solvers are applied, in [11], to a particular optimal path selection problem in the semi-conductor manufacturing domain; besides the use of Bit-Vector capabilities of the SMT solver, there are little further similarities: the problem there can be described by only one graph, having one single cost annotation for each edge (we have multiple "costs" on each node). Work bearing some similarity with ours is in [19], presenting an graph-theoretical representation of a problem in computational task parallel scheduling, and also featuring a notion of temporal separation between nodes and of resources. Only one graph is needed to represent that problem, however, and the notion of conflict there is quite different from the one presented here. Besides the example above in clock routing, Bit-Vector capabilities of SMT solvers have been employed typically (but not only) to software and hardware validation as, for example, in [12], [13], [23]. This happened before and after the remarkable complexity of Bit-Vector reasoning was emphasised in [18], confirming the practical relevance and convenience of the theory nevertheless [10]. Our work provides a further witness of this phenomenon.

Our own recent works [5,6,7] propose the use of SMT solvers for finding optimal paths along processes executed in parallel, combined with the problem of picking resources at each selected path node under the constraint of mutual resource interaction. None of them can accommodate either cyclic graphs or interactions with arity greater than 2, which is a first contribution of the present paper; in those works, we expose the issue (largely under-explored in the current literature) of ascertaining the correctness of the SMT code we use, and provide a theorem-prover based solution. As a second contribution, we will show in Section 6 that that technique is quite general, by applying it to the considered case to obtain correct-by-construction SMT code using the theorem prover Isabelle/HOL. As a final contribution, the approach presented in this paper avoids the issues entailed by more standard techniques in the field of non-linear programming (see the second paragraph of this section) by adopting a holistic attitude, in that it

expresses all the constraints and the optimality requirements for our problem as a single stack of SMT assertions, without splitting it into sub-problems.

4 Formal model of the problem and of its solution

In what follows, $\operatorname{dom} P$, $\operatorname{ran} P$ and $P[X]$ will represent, respectively, the domain, range of a relation P, and the image through P of the set $X \subseteq \operatorname{dom} P$, while 2^Y will denote the power set of a set Y. We are given n finite, simple, directed *graphs* $G_1, \ldots, G_n$; we assume their node sets to be mutually disjoint. Usually, the graphs will also be connected, but we will not assume that. Since each of the graphs G_i is simple, it can be thought of as a finite set of ordered pairs of nodes (j, k) (that is, the set representing the covering relation corresponding to G_i), each representing a directed edge from node j to node k. Therefore, we can define

$$G := \bigcup_{i=1}^{n} G_i,$$

and denote by $V(G')$ the set of nodes touched by any edge in $G' \subseteq G$, that is

$$V : 2^G \to 2^{\operatorname{dom} G \cup \operatorname{ran} G} \qquad V(G') := \bigcup_{(v,v') \in G' \subseteq G} \{v, v'\},$$

G, being a set of ordered pairs, can be regarded as the set-theoretical representation of a relation, and has, therefore, a domain and a range. The requirement for the node sets of distinct graphs to be disjoint can now be expressed as

$$\forall j \in \{1, \ldots, n\}, j' \in \{1, \ldots, n\} \setminus \{j\}. \qquad V(G_j) \cap V(G_{j'}) = \emptyset.$$

We want to describe where a given agent, who is simultaneously executing all the n graphs, is located at a given time (this agent could, for instance, be a patient being following n guidelines recommendations): the agent must start at a given node for each of the graphs, and go through the nodes of some path in each graph, spending, in each of these nodes, an amount of time which is given by a map $d : V(G) \to \mathbb{N}$. The paths will be determined from constraints we will express later on (Section 5); for the moment, we focus on how such a description can be achieved.

4.1 Paths calculation

A representation of an agent's position could be achieved, for example, by n functions (one for each graph), each defined on $\mathbb{N}$ (we assume a *discrete* representation of time) and yielding values in the node set of the relevant graph. However, as we explained in point (M2) of Section 2, we would like to exploit the discrete linear ordering naturally definable over a given path. Therefore, rather than introducing n functions yielding the node at which the executing agent is

at a given time, we introduce the auxiliary concept of *stage*: rather than directly saying that the agent, at a given time and in a given graph, is at some node, we will split this piece of information into two functions: one expressing the stage of the solution path for graph j which the agent is at time t:

$$s_j : \mathbb{N} \to \mathbb{N} \qquad\qquad j = 1 \dots n;$$

and a second one expressing the sorting of nodes belonging to the solution path for the j-th graph:

$$\eta_j : \mathbb{N} \to V(G_j) \qquad\qquad j = 1 \dots n.$$

Thus, the description of the solution paths according to the notions of stage and of time are kept separated. The function relating them is

$$\tau_j : \mathbb{N} \to \mathbb{N} \qquad\qquad j = 1 \dots n,$$

yielding, for each stage in the solution path of a given digraph $j \in \{1, \dots, n\}$, the time at which that stage starts.

Although these functions are formally defined on the whole $\mathbb{N}$, we will actually restrict their computation to given, finite natural intervals S and T of stages and times, respectively, so that the above families of functions are related by the following assertions:

$$\forall j \in \{1, \dots, n\}, \ t \in T.$$
$$s_j(t+1) = \begin{cases} s_j(t) & \text{if } t+1 < \tau_j(1 + s_j(t)) \\ 1 + s_j(t) & \text{else.} \end{cases} \tag{1}$$

$$\forall j \in \{1, \dots, n\}, \ i \in S.$$
$$\tau_j(i+1) = \tau_j(i) + d(\eta_j(i)). \tag{2}$$

The role of S and T is to put a bound on the scope of future evolution we want to follow. On one hand, this allows to reduce the search space for the SMT solver; on the other hand, the finiteness thus imposed does not prevent to consider theoretically infinite paths (e.g., when cycles are present in the graphs), reconciling their presence with the need of finite computations. One can therefore compute optimal executions for given (S_1, T_1), ending up in some configuration; then have another computational session starting from the reached configuration for some other (S_2, T_2), thus reaching another configuration, etc: although the single sessions involve finite objects, there is no bound on the number of the sessions. This implies that the computations given by (1) and (2) must assume some starting point; that is, at the smallest stage and time, the executing agent must be located in a given node $\overline{\eta_j}$ for each $j \in \{1, \dots, n\}$:

$$\forall j \in \{1, \dots, n\}. \begin{cases} \eta_j(\min S) = \overline{\eta_j} \\ \tau_j(\min S) = \min T \\ s_j(\min T) = \min S \end{cases}$$

Since S and T are not in general constant through any sequence of computational sessions, we had to define our maps (s_j, τ_j, η_j, etc) on the whole $\mathbb{N}$ rather than directly on the subsets S and T; another reason for doing so is that SMT solvers do not directly provide support for partial functions.

Finally, we must impose that η_j actually describes a walk in the directed graph G_j:

$$\forall j \in \{1, \ldots, n\}\,,\ i \in S.\ \eta_j\,(i+1) \in G_j\,[\{\eta_j\,(i)\}]\,, \tag{3}$$

which asks that the $i+1$-th node is a child (according to the digraph G_j) of the i-th node in the path selected for the digraph G_j. The notation using the square brackets was introduced at the very beginning of this section.

4.2 Interaction constraints

In Section 4.1, we have set up the problem of representing (through functions η, τ, s) a path in each G_j, and of computing the representing functions. Here we add constraints regulating how the path in distinct digraphs interact, so as to select the paths solving the earlier points (P1) and (P2) in Section 2.

We assume a finite set R of resources is given, together with maps $r_j : V\,(G_j) \to 2^R$ describing the subset of resources available to be chosen for a given node. We now want to solve the problem of picking, for each node appearing in the walks calculated in Section 4.1, resources allowed by the maps r_j (see point (P1) in the list of Section 2). Additionally, we want to do that such as to maximise the effectiveness of the picked resources along the executed path and to minimise the negative interactions between resources occurring at the same time in distinct simultaneous nodes (see point (P2) in the list of Section 2). To achieve this, we must have a database map

$$\iota : 2^R \to \mathbb{Z} \tag{4}$$

providing, for each possible combination of resources, their overall score, which is a numerical representation of how much each single resource in a given subset is effective and of how much different resources in the same subset possibly interact. Since it would be a computational waste to consider resources which cannot be prescribed in any possibly visited node, we add the constraint

$$R = \bigcup_{\substack{X \in \operatorname{ran} r_j \\ j \in \{1,\ldots,n\}}} X, \tag{5}$$

imposing that any resource must appear in $\bigcup (\operatorname{ran} r_j)$ for some digraph j. Now, the solution to the problem just introduced can be represented by maps

$$\rho_j : \mathbb{N} \to R,\ j = 1, \ldots, n$$

associating a stage number to the resource picked at that stage, for each digraph $G_1, \ldots, G_n$. The first requirement to be imposed on ρ is that the resource picked at any stage is in the permitted resources for the node corresponding to that stage:

$$\forall j \in \{1, \ldots, n\}\,,\ i \in S.\ \rho_j\,(i) \in r_j\,(\eta_j\,(i))\,, \tag{6}$$

where, as explained in Section 4.1, we limit the stages we are looking at to a finite natural interval S.

Finally, we need to optimise the interactions of resources happening at any same instant. To this end, we introduce a map $\Delta : \mathbb{N} \to 2^R$, yielding for each time the subset of all the resources picked in all the nodes happening at the given time in the selected paths across all the digraphs $G_1, \ldots, G_n$. That is, we impose

$$\Delta\,(t) := \bigcup_{j \in \{1,\ldots,n\}} \{\rho_j\,(s_j\,(t))\} \tag{7}$$

for each t in T, and then we require the SMT solver to maximise

$$\sum_{t \in T} \iota\,(\Delta\,(t))\,. \tag{8}$$

5 SMT translation

This section presents the choices we made to represent the elements expressed mathematically in Section 4 in ways amenable to an SMT solver.

For each of the formulas introduced and motivated in that section, we will choose suitable SMT-LIB code to express it, and motivate our choice. The mathematical entities (functions, sets, etc.) used in that section will be given ASCII names in order for them to be invoked in SMT-LIB code: we will make clear which ASCII name corresponds to which mathematical object as soon as it is introduced in this section. However, for the reader's convenience, table 1 summarises these correspondences.

A typical issue when translating into SMT code is to find a feasible representation, given the fact that SMT solvers are constrained to first-order logic together with a limited list of theories (e.g., integer arithmetics or Bit-Vectors). For example, sets are usually complicated to handle: while it is true that a subset of a given universe can be represented as a monadic boolean predicate over that universe, the problem is that predicates and relations are not themselves first-class objects in first-order logic. Therefore, quantifications, sets of sets, operations on sets and set-valued functions or relations are either not expressible or quickly become too complex, impacting both on performance and readability (and therefore reliability) of the code.

In our case, we do have sets of sets and operations on sets occurring in our treatment, for instance in (7). Our solution is to represent subsets of R as Bit-Vectors of length $|R|$: in this way, we can easily introduce an SMT function `resourceSelectionBV` as a counterpart of Δ:

Mathematical notation	SMT-LIB notation
Δ	`resourceSelectionBV`
ρ	`stage2ResourceSetBV`
s	`time2Stage`
$t \in T$	`isInTimeBounds t`
$i \in S$	`isInStageBounds i`
τ	`stage2StarTime`
d	`duration`
G_j	`adjMatrix_j`
η	`stage2Node`
$V\,(G_j)$	`nodeType_j`
r	`node2AllowedResources`
$\subseteq$	`isSubsetBv`

Table 1. Correspondence between mathematical and SMT-LIB names

`(declare-fun resourceSelectionBV (Int) (_ BitVec` $|R|$`))`

where $|R|$ must be replaced by its actual numerical value, since the length of a Bit-Vector cannot be a variable in SMT. This is not a problem because our model considers a fixed, finite universe of available resources (see (5)). Now, the problem of a union set operation appearing in (7) is also solved, because we can express (7) as

Listing 1.1. Assertion for (7)

```
(assert (forall ((t Int)) (=> (isInTimeBounds t)
(= (resourceSelectionBV t) (bvor
        (stage2ResourceSetBV 1 (time2Stage_1 t))
        ...
        (stage2ResourceSetBV n (time2Stage_n t)))))))
```

where `stage2ResourceSetBV`, `time2Stage` take the place of ρ and s, respectively. Moreover, `isInTimeBounds t` is the first-order logic way of saying $t \in T$, while `bvor` takes the bit-wise or of its Bit-Vector arguments (there can be any finite number of such arguments, as long as they are Bit-Vectors of the same length), effectively resulting in the union of the sets they represent. Representing an overall selection of resources at a given instant as Bit-Vector is a way of working around the limitation of fixed arity of SMT-LIB functions, in the sense that now we are allowed to specify an interaction between an arbitrary number of resources *among the* $|R|$ *available ones* by setting to one the corresponding bits in the resource selection Bit-Vector. Theoretically, the same could have been attained by describing such an interaction via a number-valued SMT function of $|R|$ boolean arguments; however, in this case the resource selection would have no longer have been a first-class object in SMT, making it the translation of some assertions (e.g., (6) and (7)) clumsier than with Bit-Vectors. An important point to be made is that the assertion above requires representing ρ as a function (`stage2ResourceSetBV`) returning a set of resources, rather than a resource. This is necessary exactly to take the union through `bvor`, and creates no problem as

long as we add the specification that $stage2ResourceSetBV$ returns a singleton. In Bit-Vector terms, this means requiring that the returned Bit-Vector has exactly one bit set to 1, which can be achieved by the following assertions (one for each digraph):

Listing 1.2. Assertion imposing that `stage2ResourceSetBV` j returns a singleton

```
(assert (forall ((i Int))
(=> (isInStageBounds i)
    (and (isPowerOfTwo (stage2ResourceSetBV j i))
    (not (= (_ bv0 |R|) (stage2ResourceSetBV j i)))))))
```

where, similarly to `isInTimeBounds`, `isInStageBounds` is the first-order logic way of requiring $i \in S$, and `isPowerOfTwo` is defined as

```
(define-fun isPowerOfTwo ((x (_ BitVec |R|))) Bool
(= (_ bv0 |R|) (bvand x (bvsub x (_ bv1 |R|)))))
```

`isPowerOfTwo` works by taking the bitwise "and" of its argument and of its argument diminished by 1. It is easy to check that this yields 0 if and only if at most one bit of the argument (assumed to have length at least two) is 1.

The remaining constraints of Section 4 are more straightforward, with (1) translating to

Listing 1.3. Assertions for (1)

```
(assert (forall ((t Int))
  (=> (isInTimeBounds (+ 1 t))
    (= (time2Stage_j (+ 1 t))
       (ite (< (+ 1 t)
               (stage2StarTime_j (+ 1 (time2Stage_j t))))
         (time2Stage_j t) (+ 1 (time2Stage_j t))))))))
```

and (2) to

```
(assert (forall ((i Int)) (=> (isInStageBounds (+ 1 i))
(= (stage2StarTime_j (+ 1 i)) (+ (stage2StarTime_j i)
(duration_j (stage2Node_j i)))))))
```

Here, `stage2StarTime` and `duration` have the roles of τ and of d, respectively. Condition (3) again features a set-theoretical aspect (the membership relation $\in$). Although in this case the formula would be simple enough to represent the relevant sets through predicates, we chose to keep using Bit-Vectors. One reason is that, by doing so, the set of nodes adjacent, in graph G_j, to a given node (appearing as $G_j(\eta_j(i))$ in (3)) becomes exactly the row for that node of the *adjacency matrix* for G_j, thus allowing us to use a well-known formalism. Therefore, we enumerate all the nodes of a given G_j through an SMT map `nodeEnum_j`, and represent the edges touching the k-th node as the k-th row of the adjacency matrix for G_j, which is a Bit-Vector. In other words, G_j gets represented, in our SMT formalism, as a function `adjMatrix_j`, of the form

```
(define-fun adjMatrix_j (nodeType_j) (_ BitVec |V(G_j)|) ... )
```

where `nodeType_j` is a finite enumeration type encompassing all the nodes of G_j. Now, formula (3) is rendered in SMT-LIB as

```
(assert (forall ((i Int)) (=> (isInStageBounds i)
(= #b1
(extractBit_j (- (nodeEnum_j (stage2Node_j (+ 1 i))) 1)
              (adjMatrix_j (stage2Node_j i)))))))
```

where `stage2Node` is the SMT name for η, and `extractBit_j k v` extracts the `k`+1-th rightmost bit of `v` as a 1-long Bit-Vector. I.e., `extractBit_j 0 v` returns the rightmost bit of `v`, `extractBit_j 1 v` returns the bit on the left of the rightmost, and so on. This behaviour can be obtained as follows:

Listing 1.4. SMT function extracting the i-th bit from a Bit-Vector of fixed length

```
(define-fun extractBit_j ((i Int)
                          (v (_ BitVec |V(G_j)|)))
                         (_ BitVec 1)
((_ extract 0 0) (bvlshr v ((_ int2bv |V(G_j)|) i))))
```

which works by shifting the bits in `v` right by `k` places, and then extracting the last bit. This is done via the SMT-LIB functions `bvlshr` and `extract`. It should be observed that one cannot directly use `extract` because its arguments must be constants. For similar reasons, we needed to define multiple copies of `extractBit`, one for each digraph, because in general the digraphs will have different numbers of nodes, while in SMT-LIB one cannot overload functions to operate on Bit-Vectors of different sizes.

Condition (6) can be easily translated into SMT-LIB thanks to the fact that we used Bit-Vectors to represent sets of resources:

Listing 1.5. Assertion for (6)

```
(assert (forall ((i Int))
  (=> (isInStageBounds i)
      (isSubsetBv (stage2ResourceSetBV j i)
         (node2AllowedResources_j (stage2Node_j i))))))
```

where `node2AllowedResources` is the SMT-LIB name of r, and the auxiliary function `isSubsetBv` encodes the relation $\subseteq$ in terms of Bit-Vectors:

```
(define-fun isSubsetBv ((x (_ BitVec |R|))
                        (X (_ BitVec |R|))) Bool
(= (bvand x (bvnot X)) (_ bv0 |R|)))
```

Note that, referring to constraint (6), we are actually requiring that $\{\rho_j(i)\} \subseteq r_j(\eta_j(i))$ in lieu of $\rho_j(i) \in r_j(\eta_j(i))$. We can do this because we already imposed that `node2AllowedResources` returns a singleton in the assertion above using `isPowerOfTwo`.

Note that the following `time2Stage`, `stage2Node`, `stage2ResourceSetBV`, and `resourceSelectionBV` completely describe the solution to our problem given by the solver. The last two represent sets as Bit-Vectors: in some occasions, it can be convenient to have a solution involving equivalent SMT objects not making use of Bit-Vectors. Such equivalent objects can be obtained from the following assertions, defining counterparts of `resourceSelectionBV` and of `stage2ResourceSetBV` expressed by using an enumeration sort `resourceSort` of cardinality $|R|$ rather than using Bit-Vectors of length $|R|$:

Listing 1.6. Assertions for Bit-Vectors-free solutions

```
(assert (forall ((t Int) (d resourceSort))
(=> (isInTimeBounds t) (= (resourceSelection t d)
(= #b1 (extractBit (- (resourceEnum d) 1)
                      (resourceSelectionBV t)))))))
(assert (forall ((g graphSort) (s Int) (d resourceSort))
(=> (isInStageBounds s) (= (stage2ResourceSet g s d)
(= #b1 (extractBit (- (resourceEnum d) 1)
                      (stage2ResourceSetBV g s)))))))
```

which must be preceded by the obvious declarations of the newly introduced functions `resourceSelection` and `stage2ResourceSet`, omitted here. `extractBit` is definied in a way analogous as that of the similar functions appearing in Listing 1.4.

6 Verification

To achieve a way of formulating our problem as an SMT problem, we have split its description and that of the solution into several first-order functions (`time2Stage`, `stage2Node`, `stage2Resource`, `node2AllowedResources`, etc). While this device made it easier to formulate the assertions for the constraints exposed in Section 4, one could wonder whether the SMT assertions faithfully reproduce the problem we started from. One way to gain confidence about this is to re-express those first-order functions inside a theorem prover and then to formally prove the correctness theorems one may want to secure. For example, it can be reassuring to show that, at each time $t \in T$, the selected resource for each graph is among the resources allowed for the node of that graph where the executing agent is at time t. We proved the following Isabelle theorem expressing this property (where `N` represents the set of graph indices $\{1, \ldots, n\}$):

Listing 1.7. A Isabelle/HOL correctness theorem

```
lemma assumes assertionUnion:
"∀ t ∈ T. resourceSelection t =
  Union {stage2Resource j (time2Stage j t)| j. j ∈ N}"
and assertionAllowed: "∀ i ∈ S . ∀ j ∈ N.
  stage2Resource j i ⊆ node2AllowedResources j (stage2Node j i)"
and assertionSomeResource: "∀ i ∈ S. ∀ j∈ N.
(stage2Resource j i) ≠ {}"
and StageInBounds:
  "∀ t∈ T. ∀ j ∈ N. time2Stage j t ∈ S"
and TimeInBounds: "tt ∈ T"
and "J ∈ N" shows
"(node2AllowedResources J (stage2Node J (time2Stage J tt))) ∩
(resourceSelection tt) ≠ {}".
```

The Isabelle formula following the keyword `shows` is the thesis of the lemma, stating the property that we just discussed, and that we wanted to grant: it says that for any chosen graph (indexed by `J`), the intersection between the resources

allowed for the visited node at time `tt` and the set of all resources selected at that time is non-empty. Note that we cannot claim, in general, that this intersection is a singleton, because there could be resources allowed in more than one node at a given time. Before the `show` keyword, there are six assumptions (the hypotheses of the lemma), each corresponding to SMT-LIB code parts. Hypothesis `assertionUnion` corresponds to assertion in listing 1.1, `assertionAllowed` to that in listing 1.5, `assertionSomeResource` is implied by assertion in listing 1.2. The theorem says that, upon the further natural assumption that S is given to be compatible with T, `resourceSelection` indeed has the expected property.

The natural objection is that what we just proved applies to an Isabelle/HOL object, and that the mere fact that we christened it with the same name as the SMT object computed in Section 5 does not fix the problem that our proof applies to the former, not to the latter. This is because the hypotheses of the Isabelle/HOL theorem above suffer from the same problem: they apply to Isabelle/HOL objects which are totally unrelated to the homonymous SMT-LIB objects introduced and computed in Section 5. The idea to overcome this issue is to formally prove that those hypotheses also apply to the SMT-LIB objects of that section. To do so, we will take advantage of the Isabelle in-built SMT-LIB generator provided by the Isabelle component *sledgehammer*, originally intended to be used for automating theorem-proving purposes (a goal we are not interested in, in this paper).

6.1 First step: generating SMT-LIB code from theorem hypotheses

We start from re-expressing the theorem in Listing 1.7 without using set-theoretical concepts. This can be done by replacing sets by boolean predicates:

```
lemma fixes
resourceSelection::"'t => 'd => bool" and
node2AllowedResources:: "('g => 'n => 'd => bool)" and
stage2ResourceSet::"'g => 's => 'd => bool" and
time2Stage :: "'g => 't => 's"
assumes assertionUnion: "∀ t d. (T t →
  (resourceSelection t d ↔
    (∃ g. (N g & stage2ResourceSet g (time2Stage g t) d))))"
and assertionAllowed: "∀ g s d.
  (S s & N g & stage2ResourceSet g s d) →
    node2AllowedResources g (stage2Node g s) d"
and assertionSomeResource: "∀ s g.
  (S s & N g) →
    (∃ d. (stage2ResourceSet g s d))"
and "∀t g. T t & N g → S (time2Stage g t)"
and "T tt" and "N gg"
shows "∃ d. ((resourceSelection tt d) &
(node2AllowedResources gg (stage2Node gg (time2Stage gg tt)) d
))"
```

The fact that Isabelle/HOL is higher-order means that it has no problem in passing from the proof for the original version of the theorem to the proof for the

new one. The reason for this translation is that this new version gets translated in computable SMT-LIB code, because it is expressed in first-order logic. To obtain this code, we simply prepend the proof of the new theorem with the line

```
sledgehammer run [provers=z3, minimize=false, timeout=1,
                  overlord=true, verbose=true] (assms)
```

This will result in automatically-generated SMT-LIB assertions, one for each hypothesis of the theorem.

6.2 Second step: linking generated SMT code to the existing code

We can now use the SMT solver to formally prove that the Isabelle-generated SMT assertions and the existing ones introduced in Section 5 are equivalent: this will imply that the correctness theorem also applies to the latter. To achieve this, we name the bodies of each assertion we are interested in: for example, let us call `unionPredicateOriginal` the conjunction of the predicates being asserted in Listings 1.1 and 1.6, and `unionPredicateIsabelle` the predicate asserted by the code automatically generated from hypothesis `assertionUnion` in the theorem above. The following assertion returning `unsat` is a proof that the Isabelle-generated assertion, for which the formal theorem above holds, is equivalent to the corresponding ones introduced in Section 5:

```
(assert(or(and unionPredicateIsabelle(not unionPredicateOriginal))
 (and (not unionPredicateIsabelle) unionPredicateOriginal)))
```

This gives a method of linking the Isabelle correctness theorem to the code from Section 5, the one effectively used for actual computations.

This method is general and can be applied to whichever correctness theorem one could desire for their SMT code (which we already did, e.g., in [5,6,7]). One aspect of this generality is that the method is not limited to a pair of SMT assertion sets (such as `unionPredicateOriginal` and `unionPredicateIsabelle` above): one could refine her SMT code, for example to improve performance, in a third set of SMT assertions (let us call it `unionPredicateOptimised`, and then show, in the same manner as done with the assertion above, that it is also equivalent to the other two assertion sets. A typical application of this consists in replacing universal quantifiers over a finite set with separate assertions, one for each instantiation of the quantified variable, usually resulting in improved performance: this can be done, for example, with most of the occurrences of `forall` in Section 5. In this particular case, the verification method ensures that the instantiation of the quantifying variable has been done correctly. We omit further details for space reasons, and further details can be found in [7].

7 Conclusions

We have presented an approach for finding optimal paths across multiple directed graphs annotated with sets of resources. An optimal path is such that it maximises the value of the resources used across the nodes it traverses (e.g., overall drug

efficacy) whilst minimising the value of resource interactions (e.g., an overall measure of the severity of drug interactions). A core feature of our approach is the possibility of enriching the constraint of interacting resources with the awareness of their temporal separation: this is represented by Δ being a function of a discrete time argument. A novelty of our work is the function ι which captures interactions between an arbitrary number of resources, and can be used to represent non-monotonic interactions between resources. For instance, we can model the case where two drugs together are problematic, but adding a further drug reduces some of the effects of the drug interaction. Another novel aspect is the presence of the bounds represented by S and T which split hard computation into sessions of limited time and stage scopes, and where the output can be used as input for the next session. In turn, this capability has permitted us to remove the limitation of acyclic graphs, typically imposed in similar work. All these features give a more realistic framework for our original application domain in healthcare.

We presented a general technique to employ rigorous formal proofs in HOL to verify first-order SMT code, and applied it to prove a particular property of our SMT encoding. A current limitation is that, since first-order logic is used in our approach as a bridge between an SMT solver and a higher-order theorem prover, the verification possibilities brought by the latter must be limited to the aspects of the SMT code which are expressible in first-order logic. In particular, we cannot currently verify the optimality aspect of our code: the optimising feature [4] is a speciality of Z3 [20] not covered by the SMT-LIB standard [3]. However, we emphasise that this limitation has limited practical impact, because typically the SMT code executed in concrete problems features a vast majority of assertions in pure SMT (i.e., expressed in the SMT-LIB standard language), and one line of code adding the requirement for optimality. It is therefore usually clear that the optimality part is correct while, in contrast, the same cannot be guaranteed for the assertions in pure SMT. Future work will explore the development of support for parallel SMT solving (e.g., using the `parallel.enable` option recently introduced in version 4.8.0 of the Z3 SMT solver).

References

1. Avriel, M., Diewert, W.E., Schaible, S., Zang, I.: Generalized concavity, vol. 63. Siam (2010)
2. Bang-Jensen, J., Gutin, G.Z.: Digraphs: theory, algorithms and applications. Springer Science & Business Media (Aug 2007)
3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK). vol. 13, p. 14 (2010)
4. Bjørner, N., Phan, A.D., Fleckenstein, L.: νz-an optimizing SMT solver. In: Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 9035, pp. 194–199. Springer (2015)

5. Bowles, J., Caminati, M.: Correct composition of dephased behavioural models. In: Proença, J., Lumpe, M. (eds.) Formal Aspects of Component Software (FACS 2017). LNCS, vol. 10487, pp. 233–250. Springer (2017)
6. Bowles, J., Caminati, M.: A flexible approach for finding optimal paths with minimal conflicts. In: International Conference on Formal Engineering Methods (ICFEM 2017). LNCS, vol. 10610, pp. 209–225. Springer (2017)
7. Bowles, J., Caminati, M.: Balancing prescriptions with constraint solvers. In: Liò, P., Zuliani, P. (eds.) Automated Reasoning for Systems Biology and Medicine. Springer (2018), In print.
8. Burer, S., Letchford, A.N.: Non-convex mixed-integer nonlinear programming: A survey. Surveys in Operations Research and Management Science **17**(2), 97–106 (2012)
9. Chartrand, G., Lesniak, L., Zhang, P.: Graphs & digraphs. Chapman and Hall/CRC (2010)
10. Dershowitz, N., Nadel, A.: Is bit-vector reasoning as hard as nexptime in practice. In: 13th International Workshop on Satisfiability Modulo Theories. Citeseer (2015)
11. Erez, A., Nadel, A.: Finding bounded path in graph using smt for automatic clock routing. In: International Conference on Computer Aided Verification. pp. 20–36. Springer (2015)
12. Falke, S., Merz, F., Sinz, C.: Llbmc: improved bounded model checking of c programs using llvm. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 623–626. Springer (2013)
13. Franzén, A., Cimatti, A., Nadel, A., Sebastiani, R., Shalev, J.: Applying smt in symbolic execution of microcode. In: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design. pp. 121–128. FMCAD Inc (2010)
14. Gabow, H.N., Maheshwari, S.N., Osterweil, L.J.: On two problems in the generation of program test paths. IEEE Transactions on Software Engineering (3), 227–231 (1976)
15. Government, S.: Polypharmacy Guidance (2nd Edition). Scottish Government Model of Care Polypharmacy Working Group (March 2015)
16. Hemmecke, R., Köppe, M., Lee, J., Weismantel, R.: Nonlinear integer programming. In: 50 Years of Integer Programming 1958-2008, pp. 561–618. Springer (2010)
17. Hughes, L., McMurdo, M.E.T., Guthrie, B.: Guidelines for people not for diseases: the challenges of applying UK clinical guidelines to people with multimorbidity. Age and Ageing **42**, 62–69 (2013)
18. Kovásznai, G., Fröhlich, A., Biere, A.: On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In: SMT@ IJCAR. pp. 44–56 (2012)
19. Lombardi, M., Milano, M., Benini, L.: Robust scheduling of task graphs under execution time uncertainty. IEEE transactions on computers **62**(1), 98–111 (2013)
20. Moura, L.D., Bjørner, N.: Z3: An efficient smt solver. In: TACAS 2008. LNCS 4963, vol. 4963, pp. 337–340. Springer (2008)
21. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic. Springer-Verlag, London, UK (2002)
22. Rossi, F., Van Beek, P., Walsh, T.: Handbook of constraint programming. Elsevier (2006)
23. Wille, R., Große, D., Haedicke, F., Drechsler, R.: Smt-based stimuli generation in the systemc verification library. In: Specification & Design Languages, 2009. FDL 2009. Forum on. pp. 1–6. IEEE (2009)