IHTC 2024 - Submission Team GIPS

Team Information

- **Name of the team**: GIPS
- **Participants**:
  - Maximilian Kratz (Real-Time Systems Lab, Technical University of Darmstadt, Germany)
  - Steffen Zschaler (Department of Informatics, King's College London, United Kingdom)
  - Jens Kosiol (Software Engineering Group, Philipps-Universität Marburg, Germany)
  - Andy Schürr (Real-Time Systems Lab, Technical University of Darmstadt, Germany)
- **Contact person**: Maximilian Kratz <maximilian.kratz@es.tu-darmstadt.de>

Description of the general GIPS approach

The Graph-Based (Mixed) Integer Linear Programming (ILP) Problem Specification (**GIPS**) Tool[1] has the goal to give developers a way of conveniently specifying and efficiently solving graph mapping problems (rule-based mapping on graphs) by using model-driven software engineering approaches. The open-source tool aims at simplifying the creation and execution of optimization problems. Users can describe their problems in a high-level specification language as graph mapping problems, from which GIPS automatically derives equivalent (M)ILP problems. Said compact specification describes the constraints and objectives of the underlying optimization problem to be solved. This approach has the benefit that developers can adopt their GIPS-based project to new requirements in a short time window without re-coding the whole (M)ILP problem generation process. It is easy for domain experts to understand the scenario when compared to manually designed (M)ILP approaches. Furthermore, it allows developers to specify (complex) (M)ILP problems with only a few lines of code (LOC) compared to thousands of LOC that would be necessary for a manual implementation. The resulting (M)ILP problems are solved, using off-the-shelf solvers, and solutions are translated back and presented to the user. Here, we briefly describe the GIPS tool and how we have applied it to the IHTC 2024 challenge.

In GIPS, optimization problems are specified starting from a graph-based representation of the key concepts and entities in the problem domain. For IHTC 2024, this includes capturing entities like *nurses* and *patients* as well as links between them (for example, to describe a *nurse* being allocated to a particular *room* in a *shift*) in a so-called metamodel. This provides a very natural way for describing problem instances easily accessible to subject matter experts. Graph Transformation (GT) rules (which are part of the high-level specification) are used to transform a given input graph, which describes in this case a given IHTC problem, into an output graph, which describes in this case a computed

[1] GIPS tool: https://gips.dev/

solution of the given IHTC problem. Figure 1 shows the GIPS process to optimize a given model. In Phase (1) a graph Pattern Matcher (PM) searches for matches of the GT rules to insert additional edges into the given input graph that represent, e.g., possible assignments of resources to tasks. The set of all possible rule applications represents the search space of the given optimization problem. In Phase (2) each possible rule application is translated into a decision variable. These decision variables combined with a set of hard and soft constraints specified in GIPSL are then translated into an (M)ILP problem, which is solved in Phase (3) by a state-of-the-art (M)ILP solver. The solution of the (M)ILP solver in the end selects a subset of all possible GT rule applications. This subset of GT rules is executed in Phase (4), which generates the needed optimized output graph.

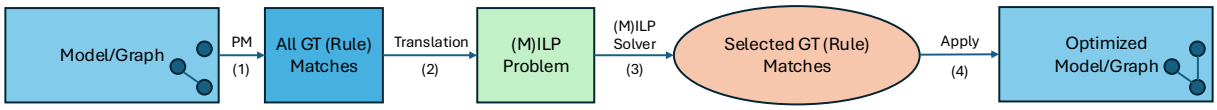

Figure 1: Concept of a GIPS run to optimize a given model.

Description of the GIPS-based solution for the IHTC 2024

Using GIPS, we implemented and evaluated a generic solution for the IHTC 2024 tasks using the following steps:

1. We specified a metamodel for the given hospital scenario using the Eclipse Modeling Framework (EMF)[2]. This can be done with the Eclipse IDE[3].

   An excerpt of our developed metamodel can be seen in Figure 2. It shows visual representations of the entities `Hospital`, `Patient`, `Room`, `Day`, and `Shift` as well as their relations. For example, a `Patient` may have several `incompatibleRooms` which must not be chosen for a valid solution, and a `Patient` can have an `assignedRoom`. Furthermore, there can be multiple `Day`s, each having potentially multiple `Shift`s. For the scheduling of *patients* and *days*, it is possible to assign an `admissionDay` to a `Patient`. Of course, the complete metamodel also features the missing entities from the IHTC 2024 problem description (e.g., `Surgeons`) and it holds the necessary attributes of all entities (for example, a `Patient` has the Boolean attribute `mandatory` that is **`true`** if a patient is mandatory).

[2] Eclipse Modeling Framework: https://projects.eclipse.org/projects/modeling.emf.emf
[3] We also provide a custom build of Eclipse with all necessary plug-ins to use GIPS with minimal effort: https://gips.dev/download/#pre-built-eclipse-application-gips

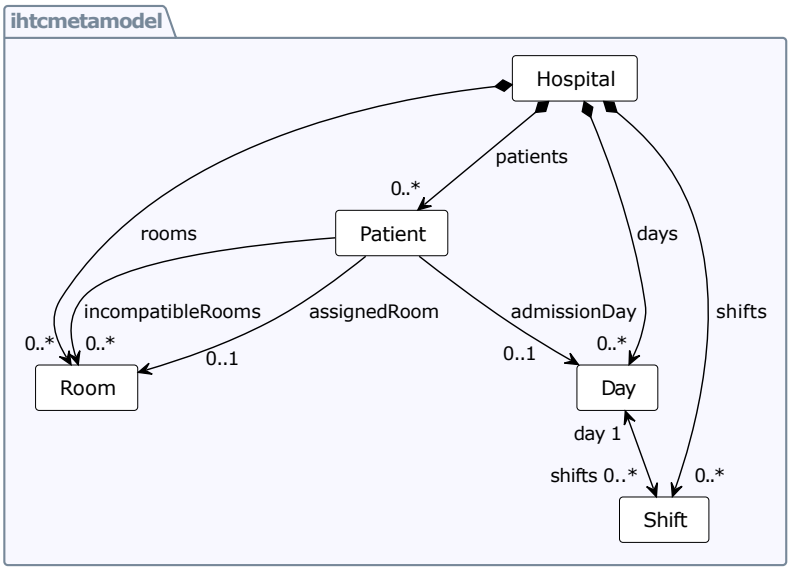

Figure 2: Excerpt of the developed metamodel to show the entities `Hospital`, `Patient`, `Room`, `Day`, and `Shift` as well as their relations.

2. To be able to import files corresponding to the custom JSON instance format, we implemented a model transformer that can transform a JSON instance file of the IHTC 2024 challenge into an EMF model conforming to our metamodel.

3. Now that we were able to load JSON instance files into GIPS, we developed multiple GIPSL specifications that can be used to solve the problem in different granularity. All specifications contain GT rules and patterns as well as additional constraints and (optional) objective functions to be optimized. GIPS compiles the high-level specifications to Java code representing an (M)ILP problem generator for the IHTC 2024. For our final submission, we use two different specifications:

   - The first specification only takes all hard constraints (H1-H6) into account.
   - The second specification takes all hard constraints (H1-H6) and some of the soft constraints (S6-S8) into account. We explain the reasons behind this in the section below.

   One of the GT rules we used in our specification is shown in Figure 3. Its purpose is to assign a patient `p` to the `admissionDay` `d`. When executed, it creates the green edge `admissionDay` between the patient `p` and the day `d`. As additional attribute constraints, it checks if the `surgeryReleaseDay` of the patient `p` is not past the day `d` and, furthermore, it checks if the `surgeryDueDate` of the patient `p` is after or at most on the day `d`.
   Similarly to the shown GT rule, other rules can be used to assign *surgeons* to *operating theaters* on specific *days*, *nurses* to *rooms* on a *shift*, etc.

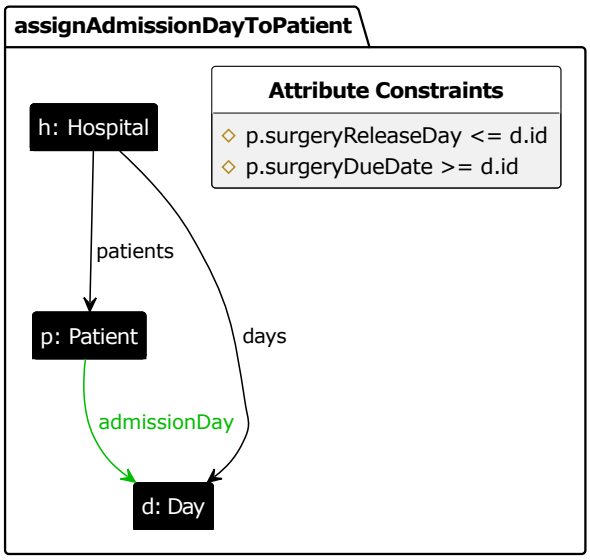

Figure 3: GT rule to assign an admission day d to a patient p while respecting p's surgery release day and their surgery due date. Upon execution of the GT rule, the green edge `admissionDay` between p and d will be created.

The following listing shows an example of a hard constraint we used in our specification that cannot be described with GT rules only. It instructs GIPS to generate a new (M)ILP constraint for every *room* that ensures the room's "load" (i.e., the number of assigned patients and occupants which is a variable in our specification) must be smaller or equal to its maximum capacity. This constraint ensures that none of the rooms will be overbooked.

```
1 // create a new constraint for every room
2 constraint with roomDayLoad {
3     // the "load" of the room must be smaller or
4     // equal to its maximum capacity
5     context.variables.load <= context.nodes.r.capacity
6 }
```

4. After the aforementioned steps, we are now able to start the GIPS runtime and load a JSON file of the competition. GIPS will automatically generate a task-specific (M)ILP problem using the previously compiled domain-specific (M)ILP problem generator.

5. After the generation step, GIPS can be instructed to solve the problem using off-the-shelf solvers like, for example, Gurobi[4] (commercial) or GLPK[5] (free and open-source). Because of its great runtime performance, we use Gurobi for tackling the IHTC 2024 tasks. The solution provided by Gurobi consists of a set of GT (rule) matches that are executed by GIPS in order to optimize

[4]Gurobi optimizer: https://www.gurobi.com/solutions/gurobi-optimizer/
[5]GLPK solver: https://www.gnu.org/software/glpk/

the input model. The result is the (still EMF-based) output model with additional edges for all assignments.

6. To be able to export a computed (EMF-based) output model to the custom JSON format required by the competition, we implemented another model transformer that can transform an EMF model to a JSON file conforming to the required solution file format.
7. Lastly, we implemented a custom CLI runner application that can be used to start and coordinate the generated GIPS code in order to automatically read an input JSON file, calculate the best-found solution within the given competition limitations, and write the solution to a JSON file.

Some additional information regarding our solution

- Due to the hard time limit of 10 minutes per instance run and the requirement to only use 4 threads, we configured the (M)ILP solver accordingly.
  - As a side note: both limitations can be specified directly in GIPSL.
  - Unfortunately, we can only guarantee the configured thread limitation for the (M)ILP solver. As we use various Java libraries in our implementation, it is possible that some of those can temporarily use more than 4 threads for parallel execution. We did not find a way to limit the parent Java process to only 4 threads.
- To accelerate our approach, we did not implement all of the soft constraints but limited the implementation to the soft constraints that produce the highest costs. In our experiments, we found that these are in most cases (sorted in descending order): *optional patients* (S8), *admission delay* (S7), and *surgeon transfer* (S6).
- For some of the larger instances, our approach that includes H1-H6 and S6-S8 is not able to find a valid solution with the given limitations. Therefore, we implemented a superior approach based on two GIPS-based stages:
  - The first stage only takes H1-H6 into account and searches for a solution that fulfills all hard constraints. This solution is typically found in less than 60 seconds for each of the given 30 competition instances.
  - After that, the second stage also takes the optional constraints S6-S8 into account and tries to find a better solution in the remaining time window. If it succeeds, the previously found solution of stage one will be overwritten. On a system similar to the computer that will be used to benchmark our solution by the organizers, the second stage is able to find a better solution in about 24 to 25 of the 30 provided cases.
- We did not implement the remaining optional constraints S1-S5 (albeit it should be possible to express them with GIPSL) because of two reasons:

1. The computational complexity of the (M)ILP approach results in a higher runtime that would violate the 10-minute limit in even more cases.
2. We ran out of time building and evaluating our approach. As we saw, the competition started in September 2024. We first heard about it a few weeks before the deadline and started working on it in mid-February. The fact that we were able to develop a largely functional solution in this limited amount of time emphasizes the rapid prototyping benefits of GIPS.

- To give some idea about the compactness of our high-level specification, we compare it to some of the generated (M)ILP problems in terms of lines of code (LOC):
  - Metamodel[6] describing types and its relations: **171 LOC**
  - GIPSL specification[7] for solving H1-H6 and S6-S8: **301 LOC**
  - Generated LP file from Gurobi for instance `i01.json`: **12854 LOC**
  - Generated LP file from Gurobi for instance `i10.json`: **199333 LOC**
  - JSON import and export are not taken into account because all approaches need some mechanism to import and export information.

Appendix 1: Detailed description of the GIPS approach

In this section, we give a more detailed description of the general GIPS approach in the context of the IHTC 2024.

GIPS is based on the Eclipse Modeling Framework (EMF) and the tool suite eMoflon::IBeX[8]. It features a domain-specific language for specifications called GIPS Language (GIPSL). GIPSL combines GT and (M)ILP constraints/objectives into one integrated framework. Constraints can be formulated using Object Constraint Language (OCL)-like expressions in GIPSL. Given a metamodel, a problem-specific (GIPSL) specification, and an instance graph, GIPS automatically derives an (M)ILP problem to optimize the instance graph according to the specification. A PM finds matches of the specified GT rules and patterns which will be transformed into said (M)ILP problem. Afterward, the included (M)ILP solver is used to solve the generated problem and decide which matches of the GT rules should be applied in order to migrate the input model to an optimal state.

More specifically, the requirements to employ GIPS for a graph mapping problem in the context of the IHTC 2024 can be explained as follows.

- All given information can be represented as an object relationship network or a graph. This is given for the IHTC 2024. (For example, every *day* holds three *shifts*, *patients* have a *due date*, *patients* have a *surgeon*, …, *nurses* have *shifts*, etc.)

[6] The metamodel can easily be created using a graphical editor in Eclipse and must not be written by hand. In this comparison, we refer to the textual representation of our EMF ecore metamodel.
[7] Please keep in mind that this specification can solve all IHTC 2024 instances and is not specific to one problem only.
[8] eMoflon::IBeX: https://emoflon.org/ibex/

- A solution to the problem is represented by a modified graph that contains additional edges that, in case of the competition, assign …
  - *patients* to *admission days*, *rooms*, and *operating theaters*,
  - *surgeons* to *operating theaters*,
  - *nurses* to *rooms*, etc.
- GT rules describe the operations required to generate these additional edges (with the consideration of local hard constraints, e.g., the enforcement of correct types (e.g., *patient*) as well as attribute constraints like "*patient* `p` is mandatory if `p.mandatory == true` holds").
- All possible application points of the aforementioned GT rules create the search space of the optimization problem. For example, *patient_1* can be assigned to *room_1*, *room_2*, …
- Global hard constraints and additional soft constraints can be formulated in GIPSL. An example of a global hard constraint could be: *"All mandatory patients must have exactly one room and exactly one admission day assigned."*
- The PM searches for all valid application points for the GT rules, which results in the search space for the optimization problem.
- A derived domain-specific generator creates an (M)ILP problem based on the set of all valid application points, all constraints, and the objective function formulated in GIPSL.
- The generated (M)ILP problem gets solved and the found solution determines on which application point(s) a GT rule must be executed. The result is an optimized graph that contains the additional edges as explained above.

Appendix 2: Technical requirements to run our GIPS-based solution

Our solution has the following technical requirements on the computer running it:

- Recent Linux like Ubuntu 24.04.
- Gurobi Optimizer v12.0.1 (currently the latest version[9]) must be installed.
  - Gurobi offers free academic licenses for research.
- Java 21 must be available[10].
- Some CLI tools must be installed for our runner script:
  - `unzip`, `rsync`
  - Standard utilities like `mv`, `rm`, `mkdir`, etc.

We are happy to provide a detailed description and a JAR file to verify our results.

If you have any questions regarding our submission or approach, feel free to contact us.

[9] Gurobi Optimizer v12.0.1: https://support.gurobi.com/hc/en-us/articles/32255882129041-Gurobi-12-0-1-released
[10] We use the Eclipse Temurin JDK: https://adoptium.net/temurin/releases/?os=linux&arch=x64&package=jdk&version=21