An Interactive Environment for Plan Visualization and Generation: InLPG[*]

Alfonso E. Gerevini and **Alessandro Saetti**

Department of Electronics for Automation, University of Brescia, Italy
{gerevini,saetti}@ing.unibs.it

Introduction

In the last years, the field of fully-automated plan generation has significantly advanced. However, in a real-world application it is likely that the plans synthesized by planning systems are not directly executable without an inspection of the user, who might require that some changes be performed in order to make the plan more suitable, robust or more preferable, or to update the problem specification, which could have evolved during the planning process. Moreover, for problems that are computationally hard, in order to effectively find a solution, it could be useful that a human interacted with the planner during search to help it.

In the literature, several mixing initiative approaches to plan generation have been proposed (e.g., (Myers *et al.* 2003; Veloso, Mulvehill, & Cox 1997)). In this work, we present a planning environment, called InLPG, supporting plan visualization and mixed-initiative plan generation, in which the user interact with the state-of-the-art planner LPG. The human knowledge and requirements that are interactively handled by InLPG concern some aspects of: the plan under construction (e.g., the request of particular start times for some actions or the addition/removal of certain actions), the planning problem under consideration (e.g., the addition/removal of a problem goal), and the plan generation process (e.g., the introduction of landmarks or the modification of choices made by the automated search heuristics).

Our system is an implementation of a more general approach aiming at: (i) providing a framework for domain-independent plan generation and visualization; (ii) supporting human interaction with a planner through some tools for effective plan visualizations and inspection; (iii) facilitating the use of planning technology to domain experts that are not planning experts; (iv) exploiting human advices during planning in order to help the planner to solve hard problems or to compute high quality solutions satisfying the user requirements.

Architecture of InLPG

The proposed framework is based on LPG, a well-known approach to efficient plan generation and adaptation (Fox *et al.* 2006; Gerevini, Saetti, & Serina 2003; 2008). LPG's plan representation is based on *linear action graphs* (Gerevini, Saetti, & Serina 2003), which are variants of the well-known planning graphs. A linear action graph (LA-graph) is a directed acyclic leveled graph alternating between a proposition level, i.e., a set of domain propositions, and an action level, i.e., one ground domain action and a set of special dummy actions, called “no-ops", each of which propagates a proposition of the previous level to the next one. If an action is in the graph, then its preconditions and positive effects appear in the corresponding proposition levels of the graph. Moreover, a pair of propositions or actions can be marked as mutually exclusive at every graph level where the pair appears. An action graph can represent a non-valid plan for the problem under consideration, since it may contain some *flaws*, i.e., an action with precondition nodes that are not *supported* (for a detailed description, see (Gerevini, Saetti, & Serina 2003)).

LPG uses a stochastic local search process that iteratively modifies the current graph until there is no flaw or a certain search limit is exceeded. LPG deals with an unsatisfied precondition by inserting into or removing from the graph a new or existing action, respectively. Starting from an initial LA-graph, a local search process transforms it into a LA-graph representing a valid plan through the iterative application of some search steps modifying the graph. The high-level search schema of LPG can be briefly described as follows:

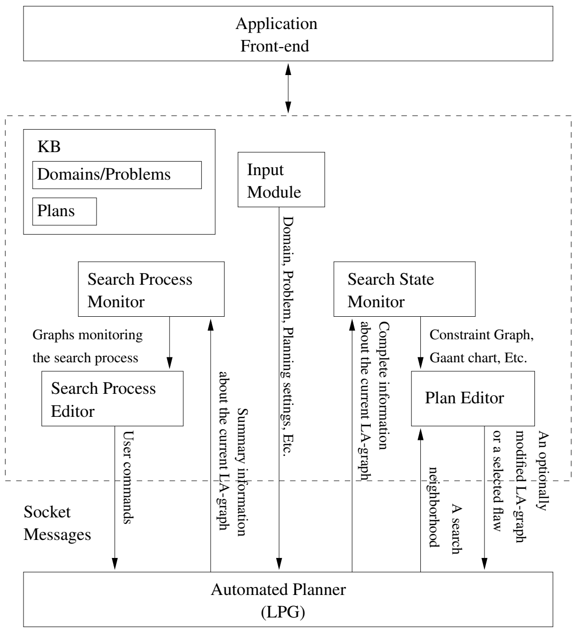
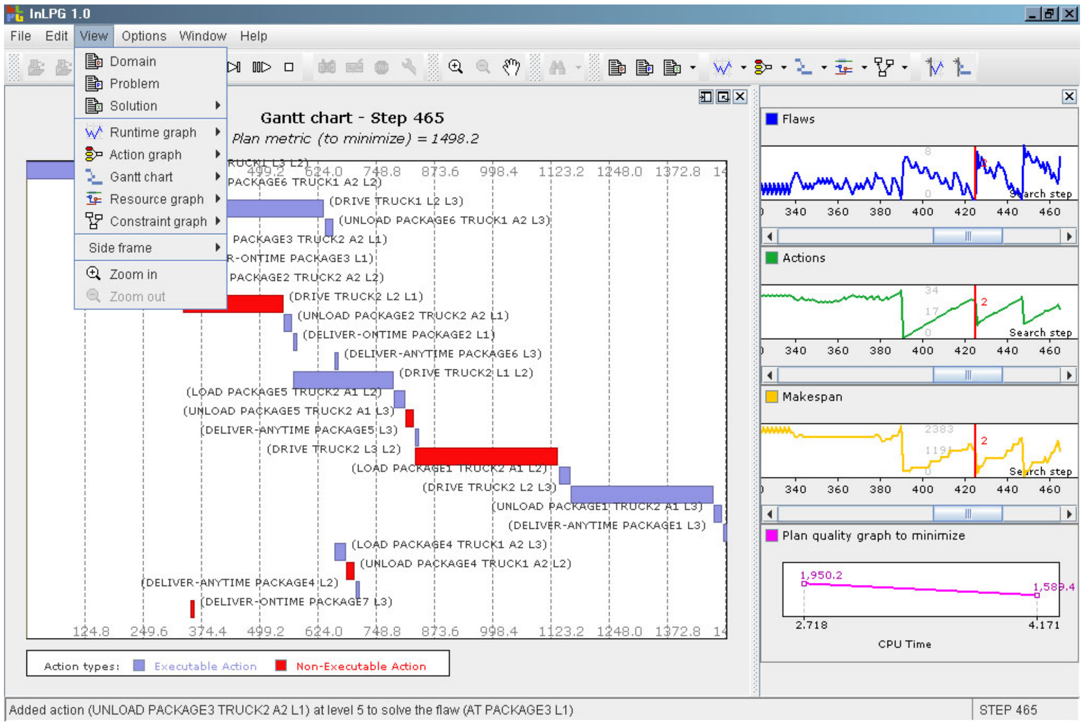

Application Front-end
KB
Domains/Problems
Plans
Input Module
Search Process Monitor
Search State Monitor
Graphs monitoring the search process
Constraint Graph, Gaant chart, Etc.
Search Process Editor
Plan Editor
Domain, Problem, Planning settings, Etc.
Summary information about the current LA-graph
User commands
Complete information about the current LA-graph
A search neighborhood
An optionally modified LA-graph or a selected flaw
Socket Messages
Automated Planner (LPG)

Figure 1: A sketch of the main components integrated in our environment and their interactions.

[*]InLPG has been developed with the help of some undergraduates. We would like to thank, in particular, Fabrizio Bonfadini.



Added action (UNLOAD PACKAGE3 TRUCK2 A2 L1) at level 5 to solve the flaw (AT PACKAGE3 L1)

Figure 2: A screenshot of the graphical user interface of InLPG.

```
1.  Set the current LA-graph $\mathcal{A}$ to the empty plan;
2.  While $\mathcal{A}$ contains a flaw or a predefined search step limit
            is not exceeded do
3.      Select a flaw $\sigma$ in $\mathcal{A}$;
4.      Identify the search neighborhood $N(\mathcal{A}, \sigma)$, i.e.,
        the set of graph modifications removing $\sigma$ from $\mathcal{A}$;
5.      Weight the elements of $N(\mathcal{A}, \sigma)$ using a heuristic $E$;
6.      Select an LA-graph $\mathcal{A}'$ in $N(\mathcal{A}, \sigma)$ and set $\mathcal{A}$ to $\mathcal{A}'$;
7.  Return $\mathcal{A}$.
```

Our environment includes an *open-controllable* version of LPG, i.e., all the decision points of the LPG's search procedure can be controlled by an external process that, in our context, is under the control of a "human planner". In particular, at each search step an user can select a plan flaw to repair (step 3), modify the definition of the search neighborhood (step 4), and select a graph modification among those that generate the elements in the search neighborhood (step 6). The decisions optionally taken by the user overwrites the decisions taken by the heuristics of LPG.

LPG runs as a separate process, and it communicates with the rest of the environment through socket messages. The architecture of InLPG, sketched in Figure 1, consists of five main components:

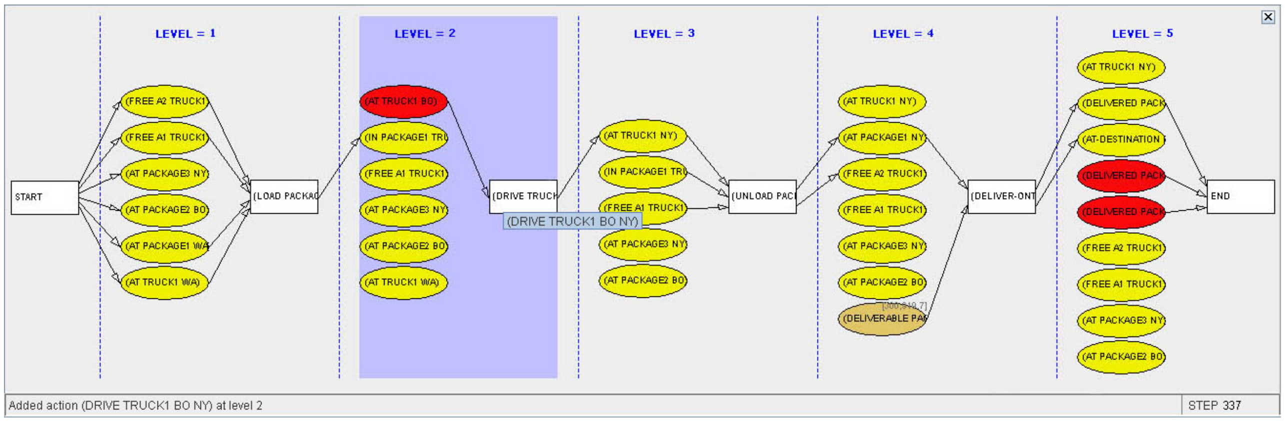
- *The Input Module*, which inputs the files containing the description of the planning problem under consideration;
- *The Search Process Monitor*, which monitors the search process, and, at each search step, displays the information about the current search state;
- *The Search State Monitor*, which provides different views of the current state during the search process;
- *The Search State Editor*, which provides tools for human driven changes to the current search state;
- *Search Process Editor*, which provides tools for human driven changes to the search process.

Figure 2 shows a screenshot of the user-interface. The left frame shows the Gantt chart of the plan computed at the 465th search step. The plan is flawed, because it contains actions that cannot be executed (dark boxes in the Gantt chart). The quality of the displayed plan is 1498.2. InLPG can visualise additional information on the LPG's search state, such as the linear action graph representing the plan under construction, a graphical representation of the ordering constraints between the actions in the plan, called *constraint graph*, and a graph showing the trend of the problem resources during the plan execution, called *resource graph*. The plots on the right frame of the screenshot of Figure 2 show, for each search step, the number of flaws, the number of actions, the makespan of the plan constructed at the corresponding search step, and, finally, the trend of the quality of all the solutions computed so far.

Walkthrough Example of an User Interaction

This section illustrates through a simple example how an user interacts with our environment for inspecting and supporting plan generation and revision in LPG.

The considered planning problem is a small problem in `Trucks` domain (Dimopoulos *et al.* 2006). The problem concerns moving three packages (`package1`, `package2` and `package3`) between New York, Washington and Boston (`NY`, `Wa` and `Bo`, respectively) by one truck (`truck1`) under



Added action (DRIVE TRUCK1 BO NY) at level 2

STEP 337

Figure 3: A portion of the screen of the graphical interface showing the LA-graph computed at the 337th search step. Square nodes are action nodes; elliptical nodes are fact nodes. Dark elliptical nodes are plan flaws (unsatisfied action preconditions). For lack of space, the label of some nodes is abbreviated. By moving the mouse on a node, a tooltip displays the corresponding full label. The darkened level is the level of the last change performed by the planning process.

certain constraints. The loading space of each truck is decomposed into a collection of areas which are organized by a spatial map imposing an order to their access and usage. In our simple problem the truck has only two areas (`a1` and `a2`), and a package can be (un)loaded onto an area `a2` of a truck only if area `a1` is free. Moreover, each package must be delivered to locations by a certain deadline indicated in the problem specification.

Figure 3 shows a snapshot of a portion of the screen containing the graphical representation of the LA-graph computed at the 377th search step, during which the user has interrupted the search process. Level 1 contains action `(load package1 truck1 a2 Wa)`, level 2 action `(drive truck1 Bo NY)`, level 3 action `(unload package1 truck1 a2 NY)`, level 4 action `(deliver-ontime package1 NY)`, finally, level 5 contains the special action ("END"), whose preconditions are the problem goals.

Figure 4 shows an example of the user interface resource window showing the trend of the fuel of `truck1` during the execution of the plan under consideration. In the initial state the fuel level of `truck1` is 0; at the end of action `(drive truck1 Bo NY)` the fuel level decreases from 0 to -7.31, since this action consumes 7.31 fuel units and the fuel level is not recharged until the end of the plan. When the line representing a certain resource crosses the grey area corresponding to the usage of the resource for a certain action, such an action is not executable in the context of the current plan. In particular, in Figure 4 action `(drive truck1 Bo NY)` is not executable because there is not enough fuel.

The effect nodes of an action that are also precondition nodes of actions at the next levels of the graph indicate the reasons why such an action is being planned. For example, action `(drive truck1 Bo NY)` is in the plan because `truck1` must be at New York before `package1` is unloaded from `truck1` to New York city (this is the activity represented by the action node at level 3). The unsupported precondition nodes of an action are plan flaws indicating

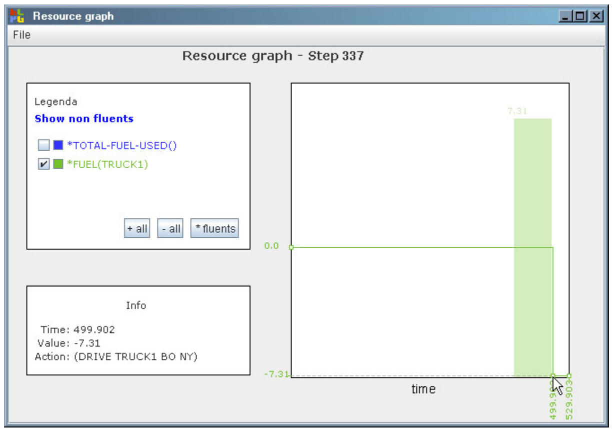
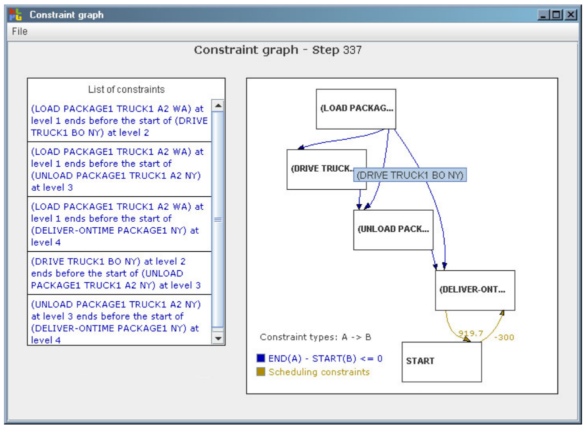

Resource graph
File
Resource graph - Step 337
Legenda
Show non fluents
☐ 🟦 *TOTAL-FUEL-USED()
☑ 🟩 *FUEL(TRUCK1)
+ all | - all | * fluents
Info
Time: 499.902
Value: -7.31
Action: (DRIVE TRUCK1 BO NY)

Figure 4: A simple example of the Resource window of the user interface for the resource `(fuel truck1)` in the running example. In the plot on the right, on the x-axis we have the plan execution time, and on the y-axis the resource level. The gray area represents the fuel level required by action `(drive truck1 Bo NY)` over all its execution.

why such an action is not executable. Action node `(drive truck1 Bo NY)` is not executable because the node labeled `(at truck1 Bo)` is not supported in the LA-graph.

Then, the user opens the window of the user interface showing the ordering constraints between the actions in the LA-graph of Figure 3 (Figure 5). The nodes in Figure 5 are actions, while edges represent action ordering constraints. For example, the edge from action `(load package1 truck1 a2 Wa)` to action `(drive truck1 Bo NY)` imposes that in the current plan `package1` is loaded from Washington on area `a2` of `truck1` before `truck1` is moved from Boston to New York. The edges connecting the start action ("START"), whose positive effects are the facts of the problem initial state, to action `(deliver-ontime package1 NY)` imposes that `package1` must be delivered



Constraint graph - Step 337

List of constraints

(LOAD PACKAGE1 TRUCK1 A2 WA) at level 1 ends before the start of (DRIVE TRUCK1 BO NY) at level 2

(LOAD PACKAGE1 TRUCK1 A2 WA) at level 1 ends before the start of (UNLOAD PACKAGE1 TRUCK1 A2 NY) at level 3

(LOAD PACKAGE1 TRUCK1 A2 WA) at level 1 ends before the start of (DELIVER-ONTIME PACKAGE1 NY) at level 4

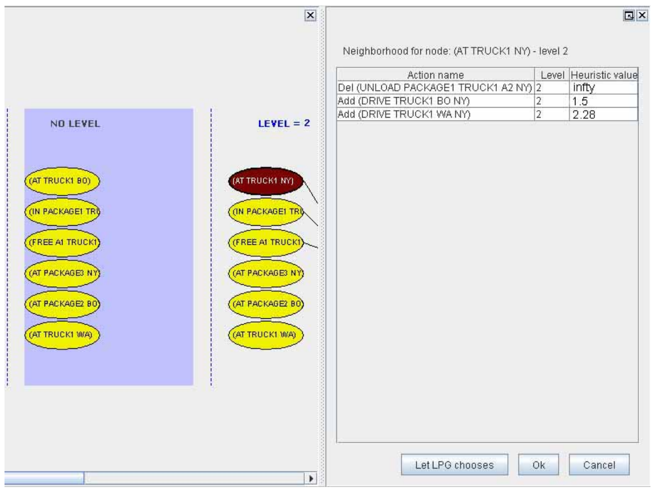
(DRIVE TRUCK1 BO NY) at level 2 ends before the start of (UNLOAD PACKAGE1 TRUCK1 A2 NY) at level 3

(UNLOAD PACKAGE1 TRUCK1 A2 NY) at level 3 ends before the start of (DELIVER-ONTIME PACKAGE1 NY) at level 4

Figure 5: A window of the graphical interface containing the constraint graph, that represents the action ordering constraints in the computed LA-graph. The label of the nodes is abbreviated. By moving the mouse on a node, a tooltip displays the corresponding full label.

to New York from 300 to 919.7 time units since the beginning of the plan.

Let us assume that, in order to deliver `package1` at New York on time, the user prefers driving `truck1` from Washington to New York instead of driving it through Boston as currently planned. Hence, the user decides to intervene at the current search step by using the tools of the search state editor for revising the current (partial) plan: she clicks the right mouse button on the box representing action `(drive truck1 Bo NY)` in the Gantt chart and, by using the context menu that is activated, she selects the option for removing such an action from the current plan. Then, she clicks the right mouse button on the box representing the flawed node `(at truck1 NY)` in the modified LA-graph $\mathcal{A}'$, and, by using the context menu that is activated, she selects the option for repairing such a flaw. The right frame shown in Figure 6 is then automatically displayed. This window indicates the possible graph modifications for repairing the selected flaw, i.e., it shows the search neighborhood of $\mathcal{A}'$ for repairing `(at truck1 NY)`. The search neighborhood is formed by three graphs obtained from $\mathcal{A}'$ by: (1) removing action node `(unload package1 truck1 a2 NY)` at level 2; (2) adding action node `(drive truck1 Wa NY)` at level 2; and (3) adding action node `(drive truck1 Bo NY)` at level 2. The flaw repair frame of the user interface also shows the heuristic values of each graph modification, i.e., the values of the heuristic evaluation function E applied to each element in the search neighborhood (see third column of the table in the right frame of Figure 6). The lower the heuristic value, the better the corresponding graph modification is. Note that since the *only* action supporting goal `(at truck1 NY)` is action `(unload package1 truck1 a2 NY)`, the heuristic value corresponding to removing this action is infinity. The heuristic value of adding action `(drive truck1 Wa NY)` is 1.5, and the heuristic value for adding action `(drive truck1 Bo NY)` is 2.28. This indicates that, if the user did not interfere in the choice



Neighborhood for node: (AT TRUCK1 NY) - level 2

Action name	Level	Heuristic value
Del (UNLOAD PACKAGE1 TRUCK1 A2 NY)	2	infty
Add (DRIVE TRUCK1 BO NY)	2	1.5
Add (DRIVE TRUCK1 WA NY)	2	2.28

Figure 6: Two frames of the user interface containing a portion of the computed LA-graph (left frame) and the search neighborhood of such a graph for repairing the flaw `(at truck1 NY)` at level 2 of the LA-graph (right frame).

of the graph modification for repairing flaw `(at truck1 NY)`, the planner would have re-inserted action `(drive truck1 Bo NY)` into $\mathcal{A}'$. On the contrary, the user selects the element in the search neighborhood corresponding to the insertion of action `(drive truck1 Wa NY)` at level 2 of $\mathcal{A}'$. Therefore, in the resulting plan `truck1` is moved to New York from Washington instead of from Boston.

Since the user also wants that in the *final* plan `truck1` arrives at New York from Washington instead of from Boston, by using the tools of the search process editor, she modifies the rest of the search process: she clicks the right mouse button on the box representing such an action in the new computed graph and, by the context menu that is activated, she selects the option imposing that the action is never removed in the rest of the search process. Finally, the user clicks on the "play" button in the tool bar of the user interface, and the automatic search process is resumed.

References

Y. Dimopoulos, A. Gerevini, P. Haslum, and A. Saetti. 2006. The benchmark domains of the deterministic part of ipc-5. In *Abstract Booklet of the competing planners of ICAPS-06*,

M. Fox, A. Gerevini, D. Long, and I. Serina. 2006. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS-06*.

A. Gerevini, A. Saetti, and I. Serina. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. *JAIR* 20:239–290.

A. Gerevini, A. Saetti, and I. Serina. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

K. Myers, P. Jarvis, M. Tyson and J. Wolverton. 2003. A mixed-initiative framework for robust plan sketching. In *Proc. of ICAPS-03*.

M. Veloso, A. Mulvehill and M. Cox. 1997. Rationale-supported mixed-initiative case-based planning. In *Proc.of IAAI-97*.