LEDA
Extension Packages

The LEDA Project

August 17, 1998

Contents

LEDA extension packages (LEPs) extend LEDA into particular application domains and areas of algorithmics not covered by the core system. LEPs satisfy the LEP requirements. It is the sole decision of the LEDA Board whether a package qualifies as a LEDA extension package.
The LEDA board advices potential contributors of LEPs to inform the board well in advance of their plans. The Algorithmic Solutions GmbH is interested in integrating successful LEPs into its spectrum of products.

1 Available Extension Packages

See Extension Packages of LEDA for the currently available packages. Users can get

- d-dimensional geometry
- dynamic graph algorithms
- graph iterators
- PQ trees

2 Extension Packages under Construction

2.0.1 Exact Linear Programming

Linear Programming is the task of opimizing a linear objective function under a set of linear constraints.

$$\max c^T x \text{ subject to } Ax \leq b.$$

Most linear programming codes use floating point arithmetic as the underlying arithmetic and hence incur rounding errors. This may invalidate the output, e.g., declare a solvable LP unsolvable or vice-versa or yielding the wrong objective value. The exact linear programming package contains an implementation of the Simplex method based on exact integer arithmetic and it contains routines that allow to check the feasibility and optimality of a basis. The package can be used to solve (small) LPs exactly and it can be used to verify the results of faster but inexact LP-solvers.

The package is under construction at the Max-Planck-Institute für Informatik (Carsten Kwappik). There is a prototype interface available for this package.

2.0.2 Approximation Algorithms for Problems on Graphs

Many problems on graphs are NP-complete. This package provides approximation algorithms for some NP-complete problems on graphs. It is under construction at the Max-Planck-Institute für Informatik (Naveen Garg).

2.0.3 Steiner Problems

Let $G = (V, E)$ be an undirected graph and let w be a non-negative cost function on the edges of G. For any subset $S \subseteq V$ of the vertices of G a Steiner tree T for S is a subtree of G connecting the vertices in S. The cost of the Steiner tree is the sum of the cost of its edges. A Steiner tree is minimal if is has minimum cost among all Steiner tree.
The Steiner tree package contains functions that compute optimal Steiner trees for small sets S and compute approximate Steiner trees any set S. The package was written by Georg Althaus at the MPII Saarbrücken.

3 The LEP Requirements

LEDA extension packages must satisfy the following requirements:

- compatibility with the LEDA philosophy.
- a LEDA-style documentation .
- the software is platform independent.
- The software is delivered as a software package which satisfies the LEP Software Packaging Guidelines.
- support by the developers for a period of at least 2 years.

The LEP integration follows these steps:

- The developer contacts the LEDA board to connect his software to LEDA as a LEP.

- The developer cooperates with a testing person who tests the software according to the LEDA test guidelines .
- The software is delivered as a software package which satisfies the LEP Software Packaging Guidelines.
- The LEDA board decides whether the package is accepted, accepted after modifications, or rejected.

4 The LEP Software Packaging Guidelines

The LEP is self-contained but integratable into the LEDA software tree. A LEP can be seen as a small software packet which is distributed over a tree structure descending from a root named after the LEP, for example `dyn_graph`. Descending from this root are the following directories:

`incl` contains the include files of the project

`src` contains the implementation files of the project

`doc` contains the documenation sources and corresponding tools to create the manual pages (if not identical to the LEDA tools)

`test` contains the test programs and test data

`demo` contains available demos for the package

We also expect files in the root dir like `Ackknowledgements`, `CHANGES`, `FIXES`, `INSTALL`, `README` and a `Makefile` which allows to build a corresponding lib and which enables the installer to distribute the headers and libs into the bigger LEDA installation.
We expect that a `make` command executed in the `dyn_graph` directory produces the LEP-lib in this directory using the LEDA version referenced from a variable `LEDAROOT` in the makefile. We want the LEP-libs to be named uniformly as `LEP_lep_abbrev`, for example `LEP_dyn_graph`.
The default location for the tree descending from `dyn_graph` is in the `LEPS`-directory existing in the LEDA tree top level (just below the normal LEDA root). Extracting it from within `LEPS` we predefine `LEDAROOT` to `../../incl` which is adaptable in case the LEP is not installed at this default place.

```
LEPS (subdir of LEDAROOT)
├── dyn_graph:  Makefile, INSTALL, etc.
│   ├── incl
│   ├── src
│   ├── doc
│   ├── test
│   └── demo
└── dd_geokernel:  Makefile, INSTALL, etc.
    ├── incl
    ├── src
    ├── doc
    ├── test
    └── demo
```

Figure 1: The LEDA extension pack tree

For the default installation we additionally provide a `make install` command (in the Makefile) which should copy the created lib to the standard library place of the global system as well as the header files to a path `$LEDAROOT/LEP/lep_abbrev`, like `$LEDAROOT/LEP/dyn_graph`. This should allow include statements like:

```
#include <LEP/dyn_graph/graph.h>
```

5 The LEDA Board

The LEDA Board consists of Kurt Mehlhorn , Stefan Näher and Christian Uhrig

Contact: leda@mpi-sb.mpg.de