Algorithm 1015: A Fast Scalable Solver for the Dense Linear (Sum) Assignment Problem

STEFAN GUTHE, TU Darmstadt, Germany and Fraunhofer IGD, Germany
DANIEL THUERCK*, NEC Laboratories, Germany

18

This document contains the instructions for building and testing the software bundle as well as instructions on how to use the software in your own project.

CCS Concepts: • **Mathematics of computing → Combinatorial algorithms**; **Combinatorial optimization**.

Additional Key Words and Phrases: Successive Shortest Path Algorithm, Parallel Processing, Epsilon Scaling

ACM Reference Format:
Stefan Guthe and Daniel Thuerck. 2021. Algorithm 1015: A Fast Scalable Solver for the Dense Linear (Sum) Assignment Problem. *ACM Trans. Math. Softw.* 47, 2, Article 18 (April 2021), 6 pages. https://doi.org/https://dl.acm.org/doi/abs/10.1145/3442348

1 Linux

This section contains the instructions for building and running under Linux. For Windows, refer to Section 2.

1.1 Requirements

The following software is required to build the source code that comes with this publication:

- CMake 3.5
- GCC
- CUDA 10 (or later, optional for the GPU build)

1.2 Build Instructions

To build the test package that was used to generate all the performance measurements in the main publication, run the following commands inside the `lap_solver` directory:

- `mkdir build`
- `cd build`
- `cmake ../gcc`
- `make`

Since the makefile is set up to compile the same code with multiple sets of defines, it is not possible to use the parallel build, e.g. `make -j4`, as this will cause the build to fail.

*Work done while at TU Darmstadt.

Authors' addresses: Stefan Guthe, stefan.guthe@tu-darmstadt.de, TU Darmstadt, Graphical Interactive Systems Group, Fraunhofer Str. 5, 64283, Darmstadt, Germany and Fraunhofer IGD, Germany; Daniel Thuerck, daniel.thuerck@neclab.eu, NEC Laboratories, Germany.

0098-3500/2021/4-ART18 $15.00
https://doi.org/https://dl.acm.org/doi/abs/10.1145/3442348

2 Windows

2.1 Requirements

The following software is required to build the source code that comes with this publication:

- Visual Studio (at least Community Edition) 2014, 2017 or 2019
- CUDA 10.2 (other versions require build files to be patched manually)

2.2 Build Instructions

To build the test package that was used to generate all the performance measurements in the main publication, the following steps are required:

- load solution from `vc14`, `vc17` or `vc19`
- press `Ctrl+Shift+B`

3 Running Tests

The following commands will re-produce the data found in the figures and tables of the paper (Figure 1, 2 & 3 require special debug builds) using Linux. For Windows, the relative path is different (`../../../../../images/`, when executing in the build directory).

- Table 1 & Figure 4:
```
./test_cpu_evaluated -table_min 1000 -table_max 32000 -sanity -random -geometric
-geometric_disjoint -random_low_rank -rank_min 1 -rank_max 8 -double -single -runs
5 -omp
./test_cpu_evaluated -table_min 1000 -table_max 128000 -sanity -random -geometric
-geometric_disjoint -random_low_rank -rank_min 1 -rank_max 8 -double -epsilon -runs
5 -omp
```
- Figure 5:
```
./test_cpu -table_min 1000 -table_max 32000 -sanity -geometric_disjoint
-random_low_rank -rank_min 1 -rank_max 8 -double -single -runs 5
./test_cpu -table_min 1000 -table_max 128000 -random -geometric -double -single
-runs 5
./test_cpu -table_min 1000 -table_max 128000 -sanity -random -geometric
-geometric_disjoint -random_low_rank -rank_min 1 -rank_max 8
-double -epsilon -runs 5
```
- Figure 6:
```
./test_cpu -memory 257698037760 -cached_min 1000 -cached_max 1024000
-geometric_cached -geometric_disjoint_cached -sanity_cached -random_low_rank_cached
-rank_min 1 -rank_max 8 -double -epsilon -runs 5 -omp
```
- Figure 7:
```
./test_gpu -memory 3221225472 -table_min 1000 -table_max 128000 -random -double
-epsilon -runs 5
./test_gpu -memory 3221225472 -cached_min 1000 -cached_max 1024000
-geometric_cached -geometric_disjoint_cached -sanity_cached -random_low_rank_cached
-rank_min 1 -rank_max 8 -double -epsilon -runs 5
```
- Figure 8:
```
./test_gpu -memory 15032385536 -table_min 1000 -table_max 128000 -random -double
-epsilon -runs 5
./test_gpu -memory 15032385536 -cached_min 1000 -cached_max 1024000
-geometric_cached -geometric_disjoint_cached -sanity_cached -random_low_rank_cached
-rank_min 1 -rank_max 8 -double -epsilon -runs 5
```
- Figure 9:

```
./test_cpu -memory 128849018880 -img ../../images/img1s.ppm -img
../../images/img2s.ppm -img ../../images/img3s.ppm -img ../../images/img4s.ppm
-img ../../images/img5s.ppm -img ../../images/img6s.ppm -img ../../images/img7s.ppm
-img ../../images/img8s.ppm -img ../../images/img9s.ppm -img
../../images/img10s.ppm -float -single
./test_cpu -memory 128849018880 -img ../../images/img1m.ppm -img
../../images/img2m.ppm -img ../../images/img3m.ppm -img ../../images/img4m.ppm
-img ../../images/img5m.ppm -img ../../images/img6m.ppm -img ../../images/img7m.ppm
-img ../../images/img8m.ppm -img ../../images/img9m.ppm -img
../../images/img10m.ppm -float -single
./test_cpu -memory 128849018880 -img ../../images/img1s.ppm -img
../../images/img2s.ppm -img ../../images/img3s.ppm -img ../../images/img4s.ppm
-img ../../images/img5s.ppm -img ../../images/img6s.ppm -img ../../images/img7s.ppm
-img ../../images/img8s.ppm -img ../../images/img9s.ppm -img
../../images/img10s.ppm -float -epsilon -omp
./test_cpu -memory 128849018880 -img ../../images/img1m.ppm -img
../../images/img2m.ppm -img ../../images/img3m.ppm -img ../../images/img4m.ppm
-img ../../images/img5m.ppm -img ../../images/img6m.ppm -img ../../images/img7m.ppm
-img ../../images/img8m.ppm -img ../../images/img9m.ppm -img
../../images/img10m.ppm -float -epsilon -omp
./test_cpu -memory 257698037760 -img ../../images/img1l.ppm -img
../../images/img2l.ppm -img ../../images/img3l.ppm -img ../../images/img4l.ppm
-img ../../images/img5l.ppm -img ../../images/img6l.ppm -img ../../images/img7l.ppm
-img ../../images/img8l.ppm -img ../../images/img9l.ppm -img
../../images/img10l.ppm -float -epsilon -omp
/test_gpu -memory 3221225472 -img ../../images/img1s.ppm -img
../../images/img2s.ppm -img ../../images/img3s.ppm -img ../../images/img4s.ppm
-img ../../images/img5s.ppm -img ../../images/img6s.ppm -img ../../images/img7s.ppm
-img ../../images/img8s.ppm -img ../../images/img9s.ppm -img
../../images/img10s.ppm -float -epsilon
/test_gpu -memory 3221225472 -img ../../images/img1m.ppm -img
../../images/img2m.ppm -img ../../images/img3m.ppm -img ../../images/img4m.ppm
-img ../../images/img5m.ppm -img ../../images/img6m.ppm -img ../../images/img7m.ppm
-img ../../images/img8m.ppm -img ../../images/img9m.ppm -img
../../images/img10m.ppm -float -epsilon
/test_gpu -memory 3221225472 -img ../../images/img1l.ppm -img
../../images/img2l.ppm -img ../../images/img3l.ppm -img ../../images/img4l.ppm
-img ../../images/img5l.ppm -img ../../images/img6l.ppm -img ../../images/img7l.ppm
-img ../../images/img8l.ppm -img ../../images/img9l.ppm -img
../../images/img10l.ppm -float -epsilon
/test_gpu -memory 15032385536 -img ../../images/img1s.ppm -img
../../images/img2s.ppm -img ../../images/img3s.ppm -img ../../images/img4s.ppm
-img ../../images/img5s.ppm -img ../../images/img6s.ppm -img ../../images/img7s.ppm
-img ../../images/img8s.ppm -img ../../images/img9s.ppm -img
../../images/img10s.ppm -float -epsilon
/test_gpu -memory 15032385536 -img ../../images/img1m.ppm -img
../../images/img2m.ppm -img ../../images/img3m.ppm -img ../../images/img4m.ppm
-img ../../images/img5m.ppm -img ../../images/img6m.ppm -img ../../images/img7m.ppm
-img ../../images/img8m.ppm -img ../../images/img9m.ppm -img
../../images/img10m.ppm -float -epsilon
/test_gpu -memory 15032385536 -img ../../images/img1l.ppm -img
../../images/img2l.ppm -img ../../images/img3l.ppm -img ../../images/img4l.ppm
```

```
-img ../../images/img5l.ppm -img ../../images/img6l.ppm -img ../../images/img7l.ppm
-img ../../images/img8l.ppm -img ../../images/img9l.ppm -img
../../images/img10l.ppm -float -epsilon
```

- Table 2: Requires Auction solver which is not included in this package.
- Table 3: Data found in other tables except for limiting the number of threads to 8

4 Own Project

In order to use the software package in your own project, you need to include the `lap.h` file after setting the desired defines in your project. To get the same behaviour as the `test_cpu` program, use the following:

```
// enable OpenMP support
#ifdef _OPENMP
#  define LAP_OPENMP
#endif
// quiet mode
#define LAP_QUIET
// increase numerical stability for non-integer costs
#define LAP_MINIMIZE_V
```

In case you would like to use GPU support, use the following as a starting point:

```
// enable CUDA support
#define LAP_CUDA
// OpenMP required for multiple devices
#define LAP_CUDA_OPENMP
// quiet mode
#define LAP_QUIET
// increase numerical stability for non-integer costs
#define LAP_MINIMIZE_V
```

4.1 High-Level Interface

The high-level interface can be found in the `test_cpu.cpp` and `test_gpu.cu` files. The CPU functions are:

```
template <class SC, class TC, class CF, class TP>
void solveAdaptiveOMP(TP &start_time, int N1, int N2, CF &get_cost, int *rowsol,
                      int entries, bool epsilon)
```

and

```
template <class SC, class TC, class CF, class TP>
void solveAdaptive(TP &start_time, int N1, int N2, CF &get_cost, int *rowsol,
                   int entries, bool epsilon)
```

Where:

- `SC` is the type used within the solver.
- `TC` is the type for storing the cost values.
- `TP &start_time` is starting time of the test returned by `std::chrono::high_resolution_clock::now();`
- `int N1` and `int N2` specify the size of the problem
- `CF &get_cost` is the cost function lambda that takes two parameters `int x, int y` ($0 \leq x < N1$ and $0 \leq y < N2$) and returns the cost of type `TC`

- `int *rowsol` points to the row solution being returned
- `bool epsilon` enables out ϵ-Pricing and should always be enabled.

The corresponding GPU functions are:

```
template <class SC, class TC, class CF, class STATE, class TP>
void solveCUDA(TP& start_time, int N1, int N2, CF& get_cost_gpu, STATE* state,
               lap::cuda::Worksharing& ws, long long max_memory, int* rowsol,
               bool epsilon, bool silent)
```

and

```
template <class SC, class TC, class CF, class TP>
void solveTableCUDA(TP& start_time, int N1, int N2, CF& get_cost_cpu,
                    lap::cuda::Worksharing& ws, long long max_memory,
                    int* rowsol, bool epsilon, bool sequential, bool pinned,
                    bool silent)
```

The difference between these functions is that the first one uses a device lambda `CF &get_cost_gpu` with parameters `(int x, int y, STATE &state)` while the second function uses a regular cpu lambda as in the cpu code above. Additional parameters are:

- A work sharing struct `ws`, constructed using `lap::cuda::Worksharing ws(int N1, int multiple, std::vector<int> &devices, int max_devices, bool silent);` with `multiple` usually set to 256 for better memory and thread alignment
- `STATE *state` is a used defined per GPU state passed to the `get_cost_gpu` function, including pointer to memory locations used inside the device lambda
- `long long max_memory` defines how much memory should be allocated on a single GPU for holding the cost values (either cache or table)
- `bool sequential` specifies if the `get_cost_cpu` lambda can only be called from a single thread
- `bool pinned` if true, the entire CPU cost table will be stored in pinned memory

4.2 Low-Level Interface

The low-level interface can be found in the `lap.h` include files. The single threaded CPU code consists of the following functions for solving the linear assignment and calculating the final costs:

```
namespace lap
{
  template <class SC, class CF, class I> void solve(
    int dim, CF &costfunc, I &iterator, int *rowsol, bool use_epsilon);
  template <class SC, class CF, class I> void solve(
    int dim, int dim2, CF &costfunc, I &iterator, int *rowsol,
    bool use_epsilon);
  template <class SC, class CF> SC cost(
    int dim, CF &costfunc, int *rowsol);
  template <class SC, class CF> SC cost(
    int dim, int dim2, CF &costfunc, int *rowsol);
}
```

The multi threaded CPU code uses the following interface:

```
namespace lap
{
  namespace omp
```

```
  {
    template <class SC, class CF, class I> void solve(
      int dim, CF &costfunc, I &iterator, int *rowsol, bool use_epsilon);
    template <class SC, class CF, class I> void solve(
      int dim, int dim2, CF &costfunc, I &iterator, int *rowsol,
      bool use_epsilon);
    template <class SC, class CF> SC cost(
      int dim, CF &costfunc, int *rowsol);
    template <class SC, class CF> SC cost(
      int dim, int dim2, CF &costfunc, int *rowsol);
  }
}
```

The GPU version of the interface is as follows:

```
namespace lap
{
  namespace cuda
  {
    template <class SC, class TC, class CF, class I> void solve(
      int dim, CF &costfunc, I &iterator, int *rowsol, bool use_epsilon);
    template <class SC, class TC, class CF, class I> void solve(
      int dim, int dim2, CF &costfunc, I &iterator, int *rowsol,
      bool use_epsilon);
    template <class SC, class TC, class CF> SC cost(
      int dim, CF &costfunc, int *rowsol, cudaStream_t stream);
    template <class SC, class TC, class CF> SC cost(
      int dim, int dim2, CF &costfunc, int *rowsol, cudaStream_t stream);
  }
}
```

Please refer to the same file for additional helper classes that can be used for the low-level interface.

Received June 2018; revised July 2019; revised June 2020; revised November 2020; accepted December 2020