MinDistance

Basic installation and user guide

May 14, 2018

MinDistance is a software package with several fast scalar, vector and parallel implementations for computing the minimum distance of a random linear code.

This guide complements the submitted paper *Fast Algorithms for the Computation of the Minimum Distance of a Random Linear Code* and includes basic setup and execution steps for the package.

The provided software package includes all the algorithms described in the paper (described in Section 3 of the manuscript) as well as vectorized and parallelized versions of the implementations (described in Section 4 of the manuscript).

Basic installation steps

Configuration and testing

The configuration of MinDistance comprises only two steps:

```
# Step 1. Configuration
./configure

# Step 2. Compilation
make
```

Vectorization support

MinDistance is ready to exploit vector units present in many modern processors. **By default, vectorization is disabled**.

To activate this functionality, use the `--enable-vectorization=yes` option in the configuration step:

```
./configure --enable-vectorization=yes
```

The configuration script will automatically detect the vector capabilities of the CPU and use the most advanced (SSE, AVX2 or AVX512) to improve performance.

Alternatively, the user can force a specific vectorization level by using the `sse`, `avx2` or `avx512` arguments for the `--enable-vectorization` option. Note that the selected extension must be provided by the CPU and the compiler in use.

The vectorized codes heavily rely on the architectural support for the POPCNT instruction. It is expected no performance gain (or even performance decrease) in the case of processors with vector support but without POPCNT support (e.g. old Intel Core 2 Duo processors). On modern processors, forcing SSE mode could produce higher performances than more modern modes (AVX2 and AVX512) for matrices where 128<=n<256. If you are going to evaluate matrices within those dimensions, you might achieve higher performances by forcing sse mode (`--enable-vectorization=sse`) instead of using auto mode (`--enable-vectorization=yes`).

Modifying `CFLAGS`

In order to pass a specific flag to the compiler (e.g. optimization flags), set the value of the `CFLAGS` environment variable prior to the configuration step, for example:

```
# Sets -O3
CFLAGS=-O3 ./configure
```

Testing correctness

After compilation, it is highly recommended to run a simple sanity check to validate the installation. A script with name `check_smallset_a.sh` is provided in the `test` folder to validate the correctness of the solutions using small matrices.

The expected output of a correct execution is:

```
sh check_smallset_a.sh

Testing file:  ../matrices/mat_smallset_a/atest001_4x4x0.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest002_5x3x5.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest003_7x3x4.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest004_15x5x7.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest005_9x4x2.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest006_10x4x3.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest007_11x4x3.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest008_12x4x2.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest009_14x7x2.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest010_15x7x3.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest011_16x7x4.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest012_17x7x3.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest013_18x7x5.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest014_40x20x6.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest015_100x70x7.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest016_94x70x5.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest017_90x63x6.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest018_84x47x8.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest019_73x47x7.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest020_90x50x9.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest021_50x20x8.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest022_30x10x7.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest023_23x10x6.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest024_100x15x33.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest025_277x15x110.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest026_100x20x28.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest027_115x20x33.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest028_154x20x50.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest029_153x20x50.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest030_600x20x244.in     Ok
Testing file:  ../matrices/mat_smallset_a/atest031_20x10x1.in     Ok
```

Usage

The name of the generated executable is `test_distance`. It can be invoked as:

`./src/test_distance <matrix_name>`

where `<matrix_name>` is the name of the generator to use.

Execution configuration

The configuration file with name `config.in` must reside in the current directory. Otherwise, the application will abort with an error message. This text file contains the parameters of the execution to launch.

A sample `config.in` is provided in the package. The following is a sample of the contents of the file:

```
4 10          Algorithm (1:bas; 2:opt; 3:sta; 4:sav; 5:sav+unr; 10:gray)
 5 4          Number of saved generators stored in RAM (only for sav algs.)
1 2 4 16 28   Number of cores used in the execution of the program
 10           Number of permutations to perform in the diagonalization (>=1).
0             Print matrices (0=no;1=yes)
```

For each row, only the first integer number is considered; the rest of the line is just ignored.

The following are some remarks about each line in the configuration file:

1. Algorithm selection:
   - **bas (1)**: Basic algorithm, described in Subsection 3.2 of the manuscript.
   - **opt (2)**: Optimized algorithm, described in Subsection 3.3 of the manuscript.
   - **sta (3)**: Stack-based algorithm, described in Subsection 3.4 of the manuscript.
   - **sav (4)**: Algorithm with saved additions, described in Subsection 3.5 of the manuscript.
   - **sav+unr (5)**: Algorithm with saved additions and unrollings, described in Subsection 3.6 of the manuscript.
   - **gray (10)**: Brute-force algorithm with Gray code based enumeration, described in Subsection 3.1 of the manuscript.
2. Number of generators that are saved to RAM (only for altogithms 4 and 5).
3. In parallel executions, number of threads to deploy and use.
4. Number of permutations to perform in the diagonalization stage (must be a positive integer).
5. Produce verbose output (print the input and output matrices).

Generators provided for testing purposes

- *mat_smallset_a*: A set of small code generators that can be employed for checking the installation. This set includes several generator dimensions and lengths. Depending on the architecture and the number of cores, the processing of all these generators usually takes a few minutes on the best algorithms.
- *mat_smallset_b*: Same as before, but the generators are a bit larger, and hence the processing can take longer.
- *mat_perf*: A set of large generators. The processing of one generator can take from several minutes to several weeks.

Input generator file format

Generators are provided as input data and follow a strict data format. Users must be sure they correctly prepare and format the input matrices following one of the supported formats, namely:

- Format 1:

```
3 7   matrix dimensions
 1 0 0 1 0 1 1
 0 1 0 1 1 0 1
 0 0 1 0 1 1 1
```

- Format 2:

```
3 7   matrix dimensions
[ [ 1, 0, 0, 1, 0, 1, 1 ],
  [ 0, 1, 0, 1, 1, 0, 1 ],
  [ 0, 0, 1, 0, 1, 1, 1 ] ]
```

The generators included in the package usually follow a common name scheme to easily identify them:

`CODENAME_NxKxDIST.in`

When DIST (distance) is unknown, a `_` character is used.

Sample execution and output

The output of a sample execution with the configuration file described above would include the following information:

Execution configuration information

Includes vectorization type, matrix information and configuration details extracted from `config.in`. A sample is shown below:

```
Vectorization scheme: AVX2
Matrix dimensions:  47 x 84
Chars in file after reading matrix:
Trailing char: ']'
Trailing char: '
'
Trailing char: '
'
End of chars in file after reading matrix.
Read input matrix. Elapsed time (s.): 0.000280

% Algorithm to apply:              4
% Number of saved generators:      5
% Number of cores:                 1
% Number of permutations:          10
% Print matrices:                  0
```

Gamma matrices creation information

Includes, if configured, information about the creation of data structures for saving combinations. A sample is shown below:

```
Creating vector of Gamma matrices.
End of creating vector of Gamma matrices.
Creating data structures for saving combinations.
Generators:  1  Combinations:         47  Required accum.mem.(MB): 0.0
Generators:  2  Combinations:       1081  Required accum.mem.(MB): 0.0
Generators:  3  Combinations:      16215  Required accum.mem.(MB): 0.3
Generators:  4  Combinations:     178365  Required accum.mem.(MB): 3.0
Generators:  5  Combinations:    1533939  Required accum.mem.(MB): 26.4
End of creating data structures for saving combinations.
```

In this example, a total memory of about 26 MB is employed for saving generators. The larger generators are processed, the larger the total accumulated memory will be. The user must check that this required space is not too large. The number of saved generators is defined in the `config.in` file.

Permutations and diagonalization

Includes information about the diagonalization stages. A sample is shown below:

```
Creating vector of Gamma matrices.
End of creating vector of Gamma matrices.

Diagonalizing permuted matrix 0
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 1
Matrix diagonalized.
Rank vector: 7 7 3
```

```
Diagonalizing permuted matrix 2
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 3
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 4
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 5
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 6
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 7
Matrix diagonalized.
Rank vector: 7 7 3

Diagonalizing permuted matrix 8
Matrix diagonalized.
Rank vector: 7 6 4

Diagonalizing permuted matrix 9
Matrix diagonalized.
Rank vector: 7 6 3

Best rank vector: 7 7 3

Erasing vector of Gamma matrices.
Finished matrix diagonalizations. Elapsed time (s.): 0.00
```

As can be see, the application tests several permutations in order to keep the one with the largest ranks in the Gamma matrices.

Detailed distance calculation process

Includes information about the iterative process to find the minimum distance. A sample is shown below:

```
MEMORY_ALIGNMENT:                   16
Dimensions of input matrix:         47 x 84
Dimensions of compacted Gamma mats: 94 x 3
Distance Loop. lowerDist: 1  upperDist: 84
kInput:            47
numGammaMatrices:  2
  Generators: 1
  numGMatProcessed: 1
  numGMatContrib:   2
    Gamma Matrix: 1 of 2
    generate_with_saved_alg
    process_prefix_with_saved_alg
    fill_structures_for_saved_data_for_1_s
    generate_with_saved_recursive with cores: 1  0 1
```

```
  generate_with_saved_1_s
  End Gamma. lower:  2  upper: 14        Elapsed time (s.): 0.02

  Gamma Matrix: 2 of 2
  generate_with_saved_alg
  process_prefix_with_saved_alg
  fill_structures_for_saved_data_for_1_s
  generate_with_saved_recursive with cores: 1  0 1
  generate_with_saved_1_s
  End Gamma. lower:  2  upper: 13        Elapsed time (s.): 0.02

End Combin.  lower:  2  upper: 13        Elapsed time (s.): 0.02

Generators: 2
numGMatProcessed: 1
numGMatContrib:   2
  Gamma Matrix: 1 of 2
  generate_with_saved_alg
  process_prefix_with_saved_alg
  fill_structures_for_saved_data_for_1_s
  generate_with_saved_recursive with cores: 1  0 2
  generate_with_saved_1_s
  End Gamma. lower:  3  upper: 11        Elapsed time (s.): 0.02

  Gamma Matrix: 2 of 2
  generate_with_saved_alg
  process_prefix_with_saved_alg
  fill_structures_for_saved_data_for_1_s
  generate_with_saved_recursive with cores: 1  0 2
  generate_with_saved_1_s
  End Gamma. lower:  3  upper: 11        Elapsed time (s.): 0.02

End Combin.  lower:  3  upper: 11        Elapsed time (s.): 0.02

Generators: 3
numGMatProcessed: 1
numGMatContrib:   2
  Gamma Matrix: 1 of 2
  generate_with_saved_alg
  process_prefix_with_saved_alg
  fill_structures_for_saved_data_for_1_s
  generate_with_saved_recursive with cores: 1  0 3
  generate_with_saved_1_s
  End Gamma. lower:  4  upper: 10        Elapsed time (s.): 0.02

  Gamma Matrix: 2 of 2
  generate_with_saved_alg
  process_prefix_with_saved_alg
  fill_structures_for_saved_data_for_1_s
  generate_with_saved_recursive with cores: 1  0 3
  generate_with_saved_1_s
  End Gamma. lower:  4  upper: 10        Elapsed time (s.): 0.02

End Combin.  lower:  4  upper: 10        Elapsed time (s.): 0.02

Generators: 4
numGMatProcessed: 1
numGMatContrib:   1
  Gamma Matrix: 1 of 2
```

```
    generate_with_saved_alg
    process_prefix_with_saved_alg
    fill_structures_for_saved_data_for_1_s
    generate_with_saved_recursive with cores: 1  0 4
    generate_with_saved_1_s
    End Gamma. lower:  5  upper: 10        Elapsed time (s.): 0.02

  End Combin.  lower:  5  upper: 10        Elapsed time (s.): 0.02

  Generators: 5
  numGMatProcessed: 1
  numGMatContrib:   1
    Gamma Matrix: 1 of 2
    generate_with_saved_alg
    process_prefix_with_saved_alg
    fill_structures_for_saved_data_for_1_s
    generate_with_saved_recursive with cores: 1  0 5
    generate_with_saved_1_s
    End Gamma. lower:  6  upper:  8        Elapsed time (s.): 0.04

  End Combin.  lower:  6  upper:  8        Elapsed time (s.): 0.04

  Generators: 6
  numGMatProcessed: 1
  numGMatContrib:   1
    Gamma Matrix: 1 of 2
    generate_with_saved_alg
    process_prefix_with_saved_alg
    generate_with_saved_recursive with cores: 1  0 6
    End Gamma. lower:  7  upper:  8        Elapsed time (s.): 0.06

  End Combin.  lower:  7  upper:  8        Elapsed time (s.): 0.06

  Generators: 7
  numGMatProcessed: 1
  numGMatContrib:   1
    Gamma Matrix: 1 of 2
    generate_with_saved_alg
    process_prefix_with_saved_alg
    generate_with_saved_recursive with cores: 1  0 7
    End Gamma. lower:  8  upper:  8        Elapsed time (s.): 0.22

  End Combin.  lower:  8  upper:  8        Elapsed time (s.): 0.22

End Distance.  lower:  8  upper:  8        Elapsed time (s.): 0.22

Erasing vector of Gamma matrices.
Erasing data structures for saving addition of combinations.
Computed distance: 8     Elapsed time (s.): 0.23



Distance of input matrix:  8
```

As can be seen, the application first processes combinations of one row (Generators: 1), then it processes combinations of two rows (Generators: 2), and so on. This process finishes when the lower bound is equal to or larger than the upper bound.