Program documentation

User Manual

Program description

This program is made to proof the validity of the induction assumption for the hypothesis proposed by Petr Gregor and Riste Škrekovski. This is done by testing all possible cases – all non-isomorphic perfect matchings of $B(2^d)$ in respect to $Q(d)$ automorphisms are generated and then the program tries to find a Hamiltonian path between them for each of these matching and each two unconnected vertices in opposite partitions.

These paths must consist exactly of initial perfect matching completed by edges from Q(d). If some path is not found, then the case is tested to be fulfilling the hypothesis.

Compilation and run

The source code is written in C++ using C++11 standard. Therefore it can be compiled by any C++11 compatible compiler, especially by GCC or Microsoft Visual Studio.

Before compilation, the compile-time settings (as described further) must be set to required values.

When launching the program, additional arguments changing the input and/or output may be passed. All perfect matchings fulfilling the settings are generated (when input file was not selected – then these matchings are loaded instead) and then the paths for all of them are searched.

Attention: The amounts of generated matchings can be really large (even more than 1 billion). Therefore the memory consumption could be really high (up to 13 GB). To improve speed of comparisons of generated matchings, base data structure of size about 0.5 GB is allocated on its start, regardless the final number of perfect matching found.

Available settings and arguments

All settings changing behaviour of the program can be found into settings.hpp. These are:

- DIMENSION:
    - The major setting, determining the dimension of hypercube for which matchings are generated and/or paths searched. Available values are from 3 to 5.
- HYPER_EDGE_CNT:
    - Determines the number of edges in generated matchings that must lie in $Q(d)$. For any different numbers the results are disjoint, and thus the program can run parallel for them (precise description in Algorithm section)
- CHECK_INPUT:
    - Has effect only if matchings are loaded from file. If enabled, input would be checked for consistency, in cost of slower loading.
- PROGRESS_INFO:
    - If enabled, information about state of calculation during generating matchings and testing paths are being given to standard output.
- GENERATED_MATCHINGS_INFO:

    - Period of information during generating matchings. Corresponds to the number of generated perfect matchings.
- GENERATED_PATHS_INFO:
    - Period of information during generating paths. Corresponds to the number of solved perfect matchings.

Moreover, it is possible to load/save matchings from/to file or to save (non-)found paths using arguments passed to the program:

- -i input_matching_file:
    - Instead of generating all perfect matchings, the program will load matchings written in the file in format v(1)->v(2) v(3)->v(4) … v(n-1)->v(n)
- -m output_matching_file:
    - If this argument is specified, all generated matchings will be stored in compressed format into given file.
- -o output_path_file:
    - All paths for matchings for which was found at least one unsolvable case would be saved into specified file.

Output interpretation

Output files:

- Perfect matchings file:
    - In this file are stored all found perfect matchings. To save space, they are compressed in the format of 64-bit integer (returned value from compress() method), in contrast to the input file.
- Found paths file:
    - In this file are stored paths for all matchings, for which was found at least one unsolved configuration (Only by default. If the secondary function is called (find_paths()), all matchings and paths will be stored).
    - Every block of output begins with the matching followed by results for all pairs of paths' endpoints in format "start->end: start->p(2)->p(3)->…->end" or "start->end: Path does not exist" if the path was not found.

Standard output:

On the standard output is written information describing phase of computation (e. g. generating matchings or finding paths). Except that, all important data for checking the hypothesis are showed:

- If any unsolvable matching is found during the finding of paths, it is tested for satisfying the hypothesis. If it fails, a description of the failure is shown. In addition to the requirements of the hypothesis, total number of failures for a single matching is counted and its type is determined (whether all edges lie in one layer or not) and this number is compared to expected number. For dimension 5 it was displayed that every type 1 matching has exactly 56 unsolvable pairs and every type 2 matching has exactly 1 unsolvable pair (except two cases when all edges from perfect matching are in $Q(d)$ – then these numbers are doubled).

- After the search is finished, final statistics is given – whether the hypothesis for tested matchings holds and then a list all matchings with at least one unsolved configuration with the numbers, in the order in which they were tested.

Code documentation

TODO!!!