

a local refinement. There are a few necessary restrictions on the use of this capability. You can use KL only if the existing partition has 2, 4 or 8 sets, and you request bisection, quadrisection or octasection respectively. The restriction to a small number of sets is necessary to avoid ambiguities about how to recurse. Also, the architecture you specify must have the same number of sets as the partition.

5. Input and output formats. Input to **Chaco** consists of one or more files, and the response to several interactive queries. Files are used to describe the graph, and if necessary to give geometric coordinates or an existing partition. The interactive input specifies the partitioning method and the number of sets you require. An additional optional file can be used to modify the values of various parameters that control algorithmic choices and output options. This functionality is discussed in §6.10.

5.1. Format of graph input files. The standard **Chaco** input is a graph, which is read from a file. Leading lines in this file that begin with the character ‘%’ or ‘#’ are considered comments and ignored. At its simplest a correct input file contains $n + 1$ uncommented lines, where n is the number of vertices in the graph. The first of these lines contains two required integers and may have a third. The first integer is the number of vertices in the graph, and the second is the number of edges. (Note that the number of edges is half of the sum of the number of neighbors of each vertex.) The remaining n lines contain neighbor lists for each vertex from 1 to n in order. These lists are just sets of integers which are separated by spaces and contain all the neighbors of a given vertex. The neighbors may be listed in any order. Note that vertices are numbered from 1 to n , not from 0 to $n - 1$. Sample graph files can be found in subdirectory “exec” under file names ending with “.graph”.

Chaco also accepts graphs with weights on vertices and/or edges. A third parameter on the first line of the input file controls input of weighted graphs. This number may have up to three digits. If the 1’s digit is nonzero, edge weights will be read. If the 10’s digit is nonzero, vertex weights will be read. And if the 100’s digit is nonzero then vertex numbers will be read, as described below.

Vertex weights should have small integer values. (To be conservative, the sum of all vertex weights should be representable as a standard integer.) If any vertex has a weight, then weights must be given for all of them. Vertex weights appear immediately before the corresponding neighbor list.

Edge weights can be any positive floating point value, but you are encouraged to make them small integers. *Kernighan–Lin and multilevel–KL will not work properly if edge weights are not integers.* If any edge is weighted, they all must be. Edge weights are included in the graph file immediately after the corresponding entry in the neighbor list.

If you have some vertices with many neighbors, it may be inconvenient to write the entire vertex data on a single line of the graph input file. You can split the data across multiple lines by using vertex numbers. The vertex number is the first value on a line containing data for that vertex. If you specify a vertex number for any vertex you must specify one for all of them, and vertices must still be entered in increasing order.

The most general form of the graph input file is illustrated below. The different types of optional entries are indicated by different styles of parenthesis. The digit on the first line which controls each type of optional entry is indicated by the same style of parenthesis.

```
% This is the format of the graph input file
Number-of-vertices  Number-of-edges  {1}[1](1)
{Vertex-number} [Vertex-weight]  neighbor1 (edge-weight1) ...
      :
```

There is one exception to this general graph format. If you are using the inertial method or one of the simple methods without Kernighan–Lin, then it is not necessary to provide a graph since the partitioner does not make any use of connectivity information. A graph file is still needed to read the number of vertices, but the remaining lines describing the edge lists can be skipped. Note however that the code will be unable to evaluate the quality of a partition or perform any of the post-processing options without edge information. Normally several measures of the partition quality are computed and printed, but this is skipped if the graph is not present.

5.2. Format of coordinate input files. If you are using the inertial method, you will need to provide geometric coordinates for all vertices. These are placed in a different file, examples of which can be found in subdirectory “exec” with names ending with “.coords”. These geometry files must have n uncommented lines, with line i containing the coordinates of vertex i . Each line must have 1, 2 or 3 real values, corresponding to a one-, two- or three-dimensional geometry. **Chaco** determines the dimensionality by looking at the number of values on the first line. Any number of comment lines can appear at the front of this file beginning with ‘%’ or ‘#’.

5.3. Format of assignment input files. As discussed in §4.4, **Chaco** can take an existing partition and modify or evaluate it in several different ways. The existing partition is read from a file using one of two possible formats. In the standard format, the top of the file has an arbitrary number of comment lines indicated by a leading ‘%’ or ‘#’. There follow as many lines in the file as vertices in the graph. Uncommented line i contains a single integer which is the set to which vertex i is assigned. Note that set assignment numbers start at zero.

The standard format can be inconvenient for parallel computing applications since the vertices owned by a particular processor can be scattered throughout the file. It can be useful to invert the standard format, having all the vertices assigned to processor 0 first, followed by all the vertices assigned to processor 1, etc. This input format can be selected by setting the parameter `IN_ASSIGN_INV` to be `TRUE` (nonzero), as described in §6.1. With this format, the file again begins with an arbitrary number of comment lines beginning with ‘%’ or ‘#’. The next line contains a single value n_0 which is the number of vertices in set 0. The following n_0 lines list the vertices in set 0. This is followed by a line containing n_1 , the number of vertices in set 1, and so on.