Contents

AA[1]
B[2]

Chapter 1

Introduction

Chapter 2

Individual nonzero assigment

Chapter 3

Partial assigment of rows and columns

In the last section we discussed a somewhat fine-grained method for assigning nonzeros to either A_r and A_c, as we make N independent choices (one for each nonzero) and reuse some of the partitioning information (namely, whether a row or a column is cut or uncut).

It is possibile, however, to take a coarser approach and decide for more than one nonzero at a time: the "hot restart" heuristic does precisely so, making choices for a subset of a row or a column of A: in particular, we do not only take into account whether a row/column is cut or uncut, but also reuse information on which subsets of rows and columns were assigned to the same partition.

The outline of this heuristic is the following:

---
(1) Compute the priority vector v

(2) Use v with the Overpainting algorithm and compute A_c and A_r

---

Algorithm 1: Hot restart

The general idea is that we compute the vector v, which is simply a permutation of the sequence of integers $0, \dots, m + n - 1$: the numbers $0, \dots, m - 1$ represent the m rows of A, while $m, \dots, m + n - 1$ represent the n columns of the matrix.

This priority vector is necessary for the true core of this heuristic, which is the overpainting algorithm and it is described as follows:

Input: Vector of priorities v (a permutation of the vector $[0, \dots, m + n - 1]$)
Output: A_r, A_c

   $A_r = A_c = \varnothing$
   for $i = m + n - 1, \dots, 0$ **do**
     if $v_i < m$ **then**
       Add the nonzeros of the row i to A_r
     else
       Add the nonzeros of the column i to A_c
     end if
   end for

Algorithm 2: Overpainting

Because every nonzero has both a row and a column, we have that each nonzero will get assigned (or "painted" of an imaginary color which corresponds either to A_r or A_c) and then re-assigned ("painted over") a second time.

It is possible also to give an alternative formulation for this algorithm in which we iterate forward through v, described as follows:

Input: Vector of priorities v (a permutation of the vector $[0, \ldots, m + n - 1]$)
Output: A_r, A_c

   $A_r = A_c = \varnothing$
   for $i = 0, \ldots, m + n - 1$ **do**
      Mark all the unmarked nonzeros as "evaluated" in row/column i
      if $v_i < m$ **then**
         Add these nonzeros to A_r
      else
         Add these nonzeros to A_c
      end if
   end for

Algorithm 3: Different formulation of Algorithm 2.

This formulation differs substantially from the previous one, as in this case every assignment to either A_r or A_c is final.

Lastly, another different formulation is possible: we could consider (again) individually each nonzero a_{ij} and look up v and see whether $i < j$ (where $<$ is to be intended as "i precedes j" and not as the comparison of the values) or the other way around; in the first case, the row has more priority and we assign a_{ij} to A_r, otherwise we assign it to A_c.

Note that, in any case, this overpainting algorithm is completely deterministic: A_r and A_c are uniquely determined by v and therefore the heuristic part of it lies entirely in the choice of this priority vector, as described in the next section.

3.1   Computation of the priority vector

At the beginning of the previous section, we claimed that the hot restart heuristic can take into account a discrete amount of information from the previous partitioning, and this happens precisely during the computation of the priority vector. As it is not known exactly which information is good to employ and which one is to be discarded, we took a structured approach and explored a wide variety of possibilities during the generation of v.

In particular, we defined several *generating schemes*, which can be summarized as any path in the directed graph shown in Figure 3.1. Each one of these generating schemes has three phases[1]:

1. **Use of previous partitioning**: as our final goal is to build an iterative algorithm, it is quite straightforward that we have to re-employ some of the information obtained with the previous iteration. In order to understand the effectiveness of such approach, a comparison with a full restart (therefore, discarding all the information) might be convenient. We distinguish between:

   - `pa` (partition aware): the set $[0, \ldots, m+n-1]$ is divided into two, uncut rows/columns before cut rows/columns (so with an higher priority). The next phases of the heuristic are performed in each of these two subsets independently, and they are never mixed again.

---

[1]the verbatim words are the labels used to represent that element in both Figure 3.1 and the Tables with the numerical results.

- **po** (partition oblivious): the set $[0, \ldots, m + n - 1]$ is kept in one piece.

2. **Sorting**: as v expresses priority, we want to have an effective way of ordering the rows/columns. A possible method is to look at the number of nonzeros and use that as criterium for sorting the rows/columns. Therefore we distinguish between:

   - **unsorted**: we keep the array from the previous phase untouched, and consider all of the row/columns as ties.

   - **sorted**: we perform a sorting, which can be either in ascending order (**asc**) or descending order (**desc**). Then, we can decide to move all row/columns that have exactly 1 nonzero at the back of the block, thus giving them the lowest priority.

     From a theoretical point of view, this is always convenient because by assigning such nonzeros to the row/column where they are not the only element, we never have communication. To denote this process we make a comparison with typography and call these row/columns *widows*: if the heuristic perform this move, we denote it as **w**, otherwise as **nw**.

3. **Tie-breaking**: we can find a finer method of arranging the elements of v, especially where, according to the previous phase, there were elements tied:

   - **simple**: no actual tie-breaking is performed and the sorted vector is left as-is.

   - **concat**: the unsorted vector is simply a concatenation of the rows and the columns: in order to obtain the vector that yields a complete assignment of the nonzeros to A_r or to A_c, we allow a choice (**row** or **col**), which represents whether this concatenation is "rows-columns" or "columns-rows".

   - **random**: the tied elements are shuffled randomly.

   - **mix**: we try to get, as much as possible, that rows and columns are equally distributed. This is a move toward balancing of the number of nonzeros of A_r and A_c.

     In order to achieve such distribution, there are two main ways: alternation (**alt**) and spread (**spr**); suppose there are k rows and $2k$ columns tied and we start with a row: with the first distribution we have that the first $2k$ elements are rows and column alternated, and then k columns $(r, c, r, c, r, c, \ldots, c, c, c, c)$, whereas with the second one we get k sequences of row, column, column $(r, c, c, r, c, c, \ldots, r, c, c)$. We still have to decide whether to start this distribution with a **row**, or with a column (**col**).

Therefore, as an example, the generating scheme **pa-sorted-w-mix-alt-row**, has to be interpreted as: using previous partitioning, sorting the rows/columns with respect to their number of nonzeros, moving all those with one nonzero at the back, and resolving ties by an alternation of rows and columns, starting with a row.
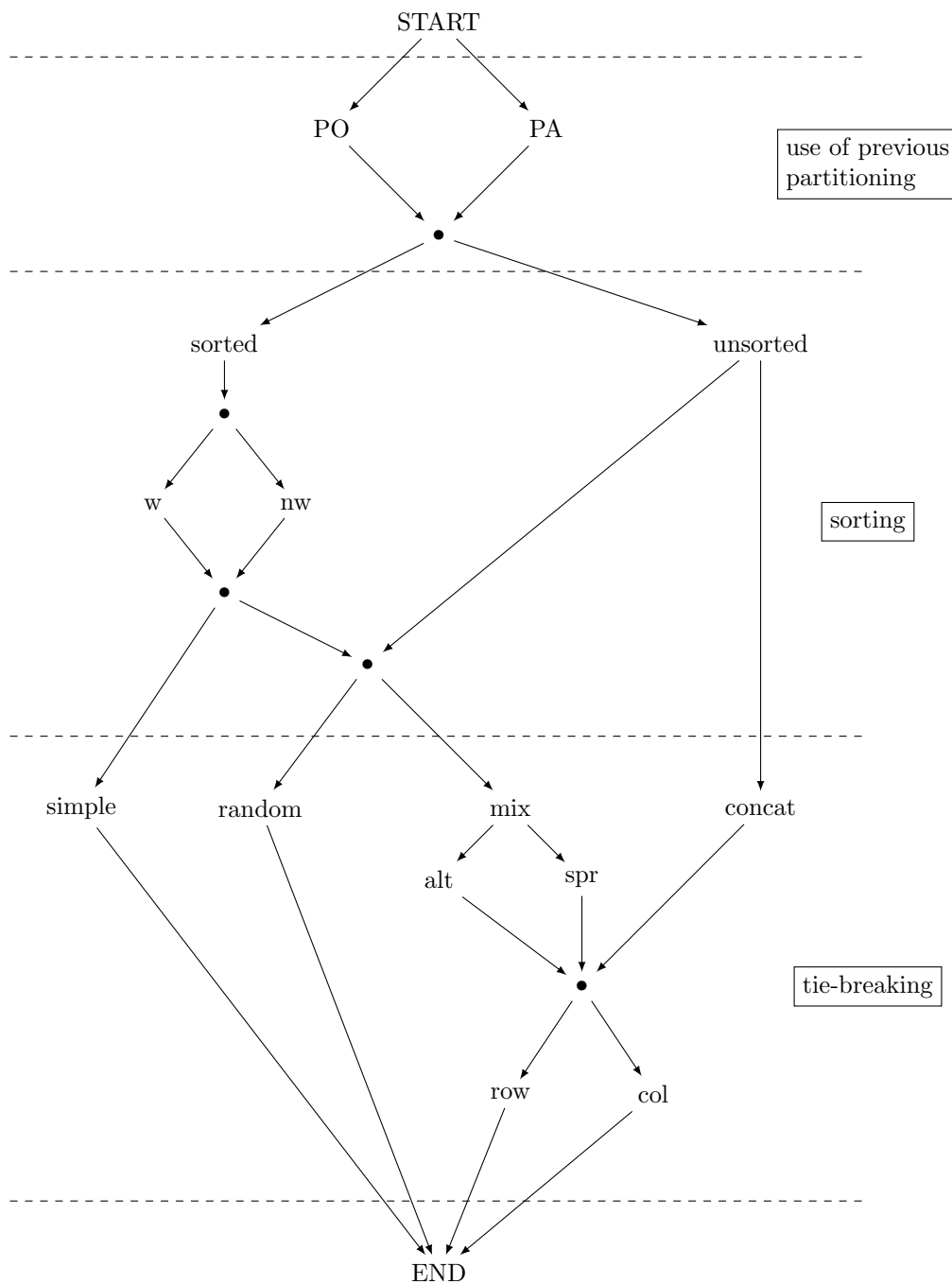
Figure 3.1: Directed graph that represents the family of heuristics used (any path from START to END). Dummy nodes (the ones without any label) were added in order to reduce the number of edges and ease legibility.

Chapter 4

Independent set formulation

4.1 Graph construction

4.2 Computation of the independent set

4.2.1 Hopcroft-Karp algorithm

Bibliography

[1] Brendan Vastenhouw and Rob H. Bisseling. "*A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication*", in: *SIAM Review* 47.1 (2005), pp. 67–95 (cited on page 1).

[2] Daniël M. Pelt and Rob H. Bisseling. "*A Medium-Grain Model for Fast 2D Bipartitioning of Sparse Matrices*", To appear. 2013 (cited on page 1).