constraint \( x_2 = x_4 \). The path-consistent constraint network version of this example is depicted in Figure 3.12(b). If we generate a path-consistent network by applying PC-1 to the original network, the algorithm’s first cycle applies REVERSE-3 to four triplets, generating the two equality constraints. A full cycle will then be executed to verify that nothing changes. This verification requires a second processing of each triplet. On the other hand, if we enforce path-consistency by PC-2, we may be able to process each triplet only once, assuming the right ordering is picked. If we apply REVERSE-3 first to \((x_1,x_3,x_2)\), that is, to the universal constraint between \(x_1\) and \(x_3\), and then to \((x_2,x_4,x_1)\), each triplet would be processed just once.

Like its arc-consistent counterpart (AC-3), PC-2 is not optimal, although we can devise an optimal algorithm, akin to AC-4. It would require operating on the relation level and maintaining supports for pairs of values. An algorithm exploiting such low-level consistency maintenance, which we will call PC-4, is available (Mohr and Henderson 1986), and its complexity bound is \(O(n^3 k^3)\) or \(O(n^3 tk)\). It is an optimal algorithm, since even verifying path-consistency has that lower bound; namely, it is \(\Omega(n^3 k^2)\).

Regarding best-case performance, we observe that PC-1, PC-2, and PC-4 have properties that parallel those of arc-consistency. Algorithms PC-1 and PC-2 can be as good as \(O(n^3 \cdot t)\) and \(O(n^3 \cdot k^2)\), respectively, while algorithm PC-4 (which was not presented explicitly) requires an order of \(O(n^3 k^3)\) (or \(O(n^3 \cdot t \cdot k)\)) even in the best case because of its initialization (see Exercise 14).

Let’s conclude our introduction to path-consistency by giving an alternative definition that may explain the origin of the term.