randomization over the \( m_1 + m_2 \) possible paths and (2) randomization, given a path, over its solutions. We initially present an algorithm, called rrLH, that adopts only a blind randomization of type (1) and, subsequently, we investigate randomization of type (2).

rrLH is described in Algorithm 1. It randomly chooses one of the paths (Step 2) and follows it (Step 4) until an NE is found; if the length (in terms of number of pivoting steps) of the path is larger than a given cutoff and there is a path that has not been visited till cutoff (Steps 5, 6, and 8), the algorithm restarts with a new path, otherwise (Step 7) cutoff is updated in an iterative deepening fashion and a restart is done.

**Algorithm 1** Lenke–Howson with random restart (rrLH)

1. cutoff = cutoff \(_0\)
2. randomly choose one path non–visited till cutoff
3. repeat
4. apply complementary pivoting
5. if the path is longer than cutoff then
6. if all the paths are visited till cutoff then
7. cutoff = cutoff + cutoff \(_0\)
8. go to Step 2
9. until a completely complementary solution is found
10. return current solution

(To improve the efficiency of the algorithm, we save the variables of the current basis before making a restart and, when a path is re–visited, we derive the last visited basis by matrix inversion and we restart from it.) The advantages of rrLH are simplicity and completeness. The potential drawbacks are that the number of possible paths is limited and that the algorithm is forced to move along fixed paths. Call \( l^* \) the shortest LH path. The compute time of rrLH is linear in \( l^* \). When \( l^* \) is known or it is possible to estimate a tight upper bound, \( cutoff \(_0\) \) can be conveniently fixed. Assigning, e.g., \( cutoff \(_0\) = l^* \), we have that the worst case compute time is \( l^* (m_1 + m_2) \), while the average is \( l^* (m_1 + m_2 + 1)/2 \). Thus, if \( l^* \) grows in length exponentially in \( m_1 + m_2 \), also rrLH compute time grows in length exponentially. Adopting a non–blind randomization of type (1) would keep the compute time, even in the best case, linear in \( l^* \) and hence is useless.

We focus on randomization of type (2). This type of randomization cannot be efficiently achieved with LH. Indeed, differently from what happens with support–enumeration algorithms [3, 4] where it is possible to start from any support profile, no every possible almost complementary basis corresponds to a feasible starting solution for LH: some sets of almost complementary variables are not feasible basic solutions, while the feasible ones can belong to LH paths or not, and in this latter case they may belong to cyclic paths that do not lead to any equilibrium (the membership of a solution to a path can be discovered only during the traversing of the path itself and therefore, if the algorithm starts from an arbitrary solution, it cannot know the path over which it is moving and cannot distinguish paths from cycles). Thus, an initial solution must be searched by using pivoting and the number of pivoting steps may be exponential in \( m_1 + m_2 \). However, we show that any algorithm with randomization of type (2) has a compute time that is asymptotically the same of our rrLH and therefore rrLH is (asymptotically) optimal in the space of the algorithms making random restarts over the LH paths. This holds even dropping completeness and considering algorithms that find an NE with a probability \( p \). Initially, we state the following lemma.

**Lemma 1.** The best cutoff of a generic randomized algorithm that finds the terminal vertex of an \( l \)–step–long path with probability \( p \) and is able to position blindly in every vertex of the path is \( l \cdot p \).

**Proof.** The best configuration of the algorithm can be obtained by minimizing the number of steps of the algorithm (i.e., \( cutoff \cdot res \), where \( res \) is the number of restarts), under the constraint that the probability to find the terminal is \( p \), i.e., \( p = 1 - (1 - cutoff/l)^{res} \), \( res \geq 1 \), and \( cutoff \geq 1 \). From \( p = 1 - (1 - cutoff/l)^{res} \) it follows that \( res = log(1 – p)/log(1 – cutoff/l) \geq 1 \), thus \( log(1 – p) \geq log(1 – cutoff/l) \), that means \( p \leq cutoff/l \). \( res \cdot cutoff \) can be rewritten as \( log(1 – p)/log(1 – cutoff/l) \cdot cutoff \). After having removed the negative constant \( log(1 – p) \) the objective is to maximize the monotonic decreasing function \( cutoff/log(1 – cutoff/l) \) under the constraint \( p \leq cutoff/l \). The optimum is \( cutoff = l \cdot p \) and \( res = 1 \).

From Lemma 1 it follows that, even when it is possible to perform a blind randomization over the solutions composing a single path, like stage (2) prescribes, the optimal configuration of the algorithm is such that this path is traversed only once without making restarts (i.e. \( res = 1 \)). Thus, this path can be safely removed from the set of the available paths at stage (1). From Lemma 1, we can easily derive the following lemma.

**Lemma 2.** The worst case compute time of a randomized algorithm finding the terminal vertex of an \( l \)–step–long path and able to position blindly in every vertex of the path is \( l \), while the average time is \( l/2 \).

Finally, we show that including randomization of type (2) the compute time keeps to be linear in \( l^* \) as stated by Lemma 3. From Lemma 2 we can state the following.

**Lemma 3.** LH with blind randomization policy of type (2) has a compute time \( O(l^*) \).

We can evaluate for specific interesting cases the ratio between the average compute time of an algorithm II adopting randomization of type (2) and the one of an algorithm II’ that does not. Assign \( cutoff = l^* \) and assume, for simplicity, that the length of all the non–shortest paths is \( l > l^* \) and \( m_1 = m_2 = m \).