```java
/*
 * タブーサーチを用いて
 * 入力グラフ (input) に対して
 * 目的関数 function を最大化するようにグラフの辺をつなぎ変える
 */
public Graph<V, E> maximize(Graph<V, E> input,
    ObjectiveFunction<Graph<V, E>> function) {
    // 現在の解とスコア
    Graph<V, E> GCurrent = input.clone();
    double fc = function.f(GCurrent);
    // 最良解とスコア
    Graph<V, E> GBest = GCurrent;
    double fBest = fc;
    // 近傍解を保存するリスト＋近傍解との差分を保存するリスト
    List<Graph<V, E>> gList = new LinkedList<Graph<V, E>>();
    List<SwapDiff> sList = new LinkedList<SwapDiff>();
    // タブーリスト
    List<SwapDiff> tabuList = new LinkedList<SwapDiff>();
    int step = 0;
    int trials = 0;
    int maxTrials = input.vertices().size();

    while(step < mSteps && trials < mSteps*maxTrials*10) {
        // 近傍解を追加
        gList.clear();
        int n = 0; // 試行回数
        while(n <= mNGList) {
            // 辺のつなぎ替えによる近傍の作成
            Graph<V, E> G1 = GCurrent.clone();
            SwapDiff sd1 = new SwapDiff(G1.rewire());
            // タブーリストに含まれていなければ解を追加
            // つなぎ替えの際に同じ辺を選んではならない
            if(!tabuList.contains(sd1)) {
                gList.add(G1);
                sList.add(sd1);
            }
            n++;
        }

        // 近傍の中で最適な解をサーチ
        Graph<V, E> GNew = gList.get(0);
        SwapDiff sNew = sList.get(0);
        double fNew = function.f(GNew);
        for(int i = 1; i < gList.size(); i++) {
            Graph<V, E> G1 = gList.get(i);
            double f = function.f(G1);
            if(f > fNew) {
                GNew = G1;
                fNew = f;
```

```
                sNew = sList.get(i);
            }
        }

        // スコアが上回っていれば最適解を更新
        if(fNew >= fBest) {
            GBest = GNew;
            fBest = fNew;
            GCurrent = GBest;
            // 最適解をタブーリストに追加
            tabuList.add(sNew);
            if(tabuList.size() > mNTabuList) {
                tabuList.remove(0);
            }
            if(verbose) System.out.println((step+1)+","+fBet);
            step++;
        }
        trials++;
    }
    return GBest;
}
```