AHC002 分析レポート

AtCoder Heuristic Contest 002 (AHC002) における kakira さんの提出と、上位3名（ats5515氏, tourist氏, tomerun氏）の提出コードを比較・解析しました。

今回のコンテストは「タイルを重複せずに通り、得点を最大化する」という典型的なパス最適化問題でした。kakira さんのコードは、**「局所的な最善を尽くす貪欲法」**であるのに対し、上位陣は**「時間をフルに使った反復改善（山登り法・焼きなまし法）」**を採用している点が決定的な差となっています。

提出タイムラインとパフォーマンス分析

ユーザー	実行時間	方針	特徴
:---	:---	:---	:---
kakira	12 ms	貪欲法 + 短手読みDFS	渦巻き状に進む固定戦略。実行時間が極端に短い。
ats5515 (1位)	1959 ms	初期解DFS + 焼きなまし法	パスの一部を切り取ってDFSで繋ぎ直す改善を繰り返す。
tourist (2位)	1961 ms	全探索DFS + 焼きなまし法	探索順序を入れ替えながら初期解を作り、局所探索で改善。
tomerun (3位)	1992 ms	多始点貪欲 + 焼きなまし法	ランダムウォークによる評価を用いた貪欲解を焼きなましで改善。

全体的な考察

kakira さんの実行時間が **12ms** であるのに対し、上位陣は制限時間（2000ms）ギリギリの **1900ms超** まで粘っています。ヒューリスティックコンテストにおいて、実行時間の余りは「スコアを伸ばす機会の損失」を意味します。

方針・実装レベルの詳細比較

1. 思考のフレームワーク：貪欲か、反復改善か

kakira さんのアプローチ

kakira さんは「今いる場所から k 手先までを見て、最も良い方向に進む」という戦略をとっています。

```
// kakiraさんの実装: 12手先まで読んで進む方向を決める
while(1) {
    clear_ndfs_state();
    n_dfs(y, x, 12, (DIR)current_dir); // 12手先読み
    if (cand_path.size() == 0) {
        // 行き止まったら方向転換
        current_dir = (current_dir + 1) % 4;
        continue;
    }
    // 決定したパスを3歩だけ進めて確定させる
    for (int i = 0; i < 3; i++) { ... }
}
```

この方針は「一度決めた移動を後から修正できない」ため、序盤の移動が終盤の自由度を奪い、スコアが頭打ちになります。

上位勢（ats5515氏）のアプローチ

1位の ats5515 氏は、**「一度作ったパスを壊して作り直す」**山登り法（焼きなまし法）を採用しています。

```
// ats5515氏の実装: パスの一部を切り取って繋ぎ直す
void SA(double timelimit) {
    while (true) {
        ti = timer.get();
        if (ti > timelimit) break; // 時間いっぱいまで繰り返す

        // パスの中からランダムな区間 [a, b] を選ぶ
        int W = rnd.nextInt((int)(1 + res.size() * 0.1 * (0.01 + remti)));
        int a = rnd.nextInt(res.size() - W);
        int b = a + W;

        // 区間 [a, b] を一旦削除し、別のルートで a と b を繋げないか試行
        // スコアが上がれば採用、上がらなければ元に戻す
        dfs2(res[a]);
        ...
    }
}
```

アドバイス:
ヒューリスティック問題では「最初から完璧なパスを作る」のは不可能です。「適当なパスをまず作り、制限時間いっぱいまでその一部をランダムに書き換えて改善し続ける」というメタヒューリスティクスの導入が必須です。

2. 時間管理（Time Management）

上位3名は全員、`get_time()` や `Timer` クラスを自作し、ループの終了条件に組み込んでいます。

- **tourist氏:** `if (get_time() > limit) break;`
- **tomerun氏:** `while Time.utc.to_unix_ms < @timelimit`

kakira さんのコードには時間による制御がなく、一瞬で計算が終わっています。
アドバイス:
`std::chrono` などを用いて、「残り時間が 1.9 秒になるまでループを回す」という構造をテンプレート化しておきましょう。

3. 評価関数の工夫

kakira さんの評価

`n_dfs` 内で「指定した方向にどれだけ進めたか」を評価していますが、タイルの得点 $P_{i,j}$ をあまり活用できていません。

tomerun 氏の評価

3位の tomerun 氏は、貪欲法で進む方向を決める際、**「ランダムウォーク」**を数回行い、その平均スコアが高い方向を選んでいます。

```
# tomerun氏の実装: 候補の各方向に対してランダムウォークで期待値を計算
dead_check_count.times do
    ns = random_walk(nr, nc, dead_check_length) + @p[nr][nc]
    if ns > best_score
        best_score = ns
        dir = d
    end
end
```

また、評価関数に「中心からの距離」などのバイアスをかけることで、外周から埋めていくような挙動を誘発させています。

---

kakira さんへの具体的な添削・改善案

1. `n_dfs` を「初期解生成」に使い、その後に「改善ループ」を入れる

現在の `n_dfs` を使った貪欲法でまず1つのパス（`vector<int> path`）を作ります。その後、以下のようなループを追加してください。

```
// 改善案のイメージ
auto start_time = chrono::system_clock::now();
while (true) {
    auto now = chrono::system_clock::now();
    if (chrono::duration_cast<chrono::milliseconds>(now - start_time).count() > 1900) break;

    // 1. パスからランダムに2点 A, B を選ぶ
    // 2. AからBまでの現在の経路を消す
    // 3. AからBまで、別の経路（DFSやランダムウォーク）で繋ぎ直す
    // 4. 新しい経路の方がスコアが高ければ更新
}
```

2. 探索の多様性を出す

kakira さんのコードは `current_dir = LEFT` から始まるなど、挙動が固定的です。 上位陣は `order`（移動方向の優先順位）を `shuffle` したり、 `next_permutation` で全パターン試したりしています。

- **tourist氏の工夫:**

```
    vector<int> order(4);
    iota(order.begin(), order.end(), 0);
    do {
        Dfs(si * n + sj); // 移動順序（URDLなど）を全通り試す
    } while (next_permutation(order.begin(), order.end()));
```

このように、同じアルゴリズムでも「探索順序を変えて何度も試す」だけでスコアは上がります。

3. タイル情報の持ち方の改善

kakira さんは `used[ID[i][j]]` でタイル使用済み判定を行っていますが、これは正しいです。 さらに高速化するために、ats5515 氏のように `bitset<1500> bs;` を使うと、メモリ効率とアクセス速度が向上し、改善ループの回数を稼ぐことができます。

総合アドバイス

- **「2秒間使い切る」ことを絶対ルールにする:**

ヒューリスティックでは、10msで終わるコードは「何も考えていない」のと同義です。山登り法を実装し、ループ回数を最大化してください。

- **「破壊と再生」を実装する:**

今の kakira さんの DFS 実装力があれば、「パスの途中を切り取って繋ぎ直す」関数はすぐに書けるはずです。これができるだけで、AHC の順位は劇的に上がります。

- **初期解は適当で良い:**

上位勢も、最初の解は単純な貪欲や DFS です。大事なのはその後の「数万回の微修正」です。

kakira さんの DFS の実装（`n_dfs`）は非常に綺麗に書けています。この「探索する力」を「改善する力」に転用できれば、上位入賞も十分に狙えるはずです。