メタヒューリスティクスの適用を効率的に記述するドメイン特化言語

2131113　冨井 陸矢　　成見研究室

1　背景

近年多くの企業や研究機関が，研究内容をモデル化した問題をコンテストとして出題している．それらのコンテストの中で生み出されたプログラムは，そのまま研究成果として学会で発表されたり [1]，実際にソフトウェアに用いられることもある．また，問題の適切なモデル化が行われることで，それまで数理最適化に馴染みのなかった人間でも，新たに参入がしやすくなっている．非厳密に数理最適化問題を解く際に用いられるアルゴリズムの枠組み（メタヒューリスティクス）はいくつかのパターンに絞られ，特に焼きなまし法 [2] 等の局所探索法とビームサーチ等の逐次探索法に分類できる．

2　問題点

2.1　プログラムの再利用性の低さ

アルゴリズムの間には共通項が多く存在するにもかかわらず，要素が整理して抽象化されておらず，プログラムの再利用が難しい．例えば焼きなまし法のプログラムを実装した後に，同じ局所探索法であるタブーサーチを試したいとなっても，大部分を書き換えることになってしまう．

2.2　高速化に伴う煩雑さ

プログラムを高速にして性能を高めるための典型的な高速化手法がいくつか存在するが，それらはいずれもプログラムを煩雑にするものである．

解の一部を改変した際に，解の評価値がどのように変わるかを調べる必要がある．これは毎回評価関数を一から計算し直すのではなく，差分計算することで高速になるが，そうすると機械的で複雑なコードとなる．

また，ビームサーチやタブーサーチにおいては 2 つの解の同一性を判定する必要がある．解の内部をすべて比較するのは時にコストの大きい処理であるため，解に対してハッシュ関数を定義し，ハッシュ値を用い判定することで高速化をすることが多い．しかし，複数のデータ構造を複合した適切なハッシュ関数の設計は容易ではなく，状態への操作の度にハッシュ値の更新の記述が必要となる．

また，一命令で複数のデータを操作できる SIMD 命令を利用すると，大幅な高速化が行えることがある．しかし，C++ の組み込み関数等を直書きするプログラミングは実装が複雑であり，可読性や保守性を大きく損なう．

3　目的

数理最適化のプログラミングにおいて生じる，上で述べたプログラムの再利用性が低く，高速化に伴うプログラミングの煩雑さの問題を解決する．そのために本研究では，いくつかの典型的なメタヒューリスティクスを適切に抽象化し，それらを利用するプログラムを実行性能を保ちながら最小限のコードで記述して，これを C++ へ変換するドメイン特化言語 OpTwoFast を設計し開発する．

```
1  TSP_SA := class(SAState<Pair>){
2    order:vector<int>;
3    init := { order = range(0,n); };
4    ?score := { // スコア関数を宣言的に記述
5      (0..(n-1)).sum(d: i =>
6      dist(points[order[x:i]],points[order[y:i+1]]));
7    };
8    get_transition := { rand_pair(1,n-1); };
9    accept_transition := (tr) => {
10       // 変化のアノテーション
11       @score(d(tr.0-1):y->tr.1, d(tr.1):x->tr.0);
12       reverse(order,tr.0,tr.1);
13   };
14 }
```

図 1　OpTwoFast による焼きなまし法の適用

4　方針

OpTwoFast プログラムは C++ のプログラムへ変換し，それを GCC 等の既存のコンパイラでコンパイルして実行可能にする．このような方針を採る利点として，C++ の豊富な標準ライブラリをそのまま使用でき，一般的なコンパイラ最適化を自力で実装せずとも優れた実行性能を得られることが挙げられる．さらに，コンテストとしての数理最適化では C++ がデファクトスタンダードとなっているのも大きな利点である．

再利用性を高めるために，各種メタヒューリスティクスについて，様々な問題へ適用する上での必要な要素を括り出し，それらを実装することにより簡単にメタヒューリスティクスを適用できるような標準ライブラリを設計する．

設計としては，宣言的な記述を用いて，処理系が関数の内容を構造化された形式で得られるようにする．また，アノテーションにより，最小限の記述から評価値の差分計算やハッシュ関数を自動的に生成する．さらに，SIMD 命令を用いて，複数の状態を並列に操作するメタヒューリスティクスへ変換できるようにする．最終的に最も良い解を選ぶことで，得られる解の質を向上させる．

5　設計

OpTwoFast により巡回セールスマン問題に焼きなまし法を適用するプログラム例を図 1 に示す．

5.1　メタヒューリスティクスの抽象化

局所探索法のアルゴリズムは，「評価値をどう計算するか」「解にもたらす遷移の型が何か」「どのように遷移を得るか」等の要素に分解され，これは焼きなまし法やタブーサーチ等で共通する．さらに，逐次探索法についても，遷

表 1　巡回セールスマン問題を解くプログラムの評価

	C++		OpTwoFast	
	行数	実行時間	行数	実行時間
焼きなまし法	25	688 ms	12	693 ms
タブーサーチ	40(13)	601 ms	12	623 ms
ビームサーチ	42(19)	253 ms	18	257 ms
chokudai サーチ	42(19)	381 ms	18	398 ms

移を抽出するか，列挙するかという違いのみで局所探索法と統一することができる．OpTwoFast では，これらの間でプログラムを書き換えるコストを小さくしている．例えば焼きなまし法は，SAState というライブラリのクラスとして提供し，これを継承して各要素を実装する．

5.2　評価値の差分計算

図 1 の 4 行目の ?score という特殊な関数により，評価関数を構造化された形式で記述する．それと対応させる形で，11 行目では @score によって変化箇所をコンパイラに指示している．これらの情報から，コンパイラは煩雑であった差分計算処理を自動的に生成する．

5.3　ハッシュ関数の生成とハッシュ値の更新

次の例のようにメンバ変数の宣言の先頭に # を加えることで，それらを用いたハッシュ関数を自動生成する．

```
TSP_BS := class(BSState<int>){
    order:vector<int>;
    #visited:vector<bool>;
    #now:int;
```

この例では，TSP_BS クラスは変数 visited と now の組により比較される．また，コンパイラは対象の変数への操作の前後に，ハッシュ値の更新処理を自動的に挿入する．

5.4　SIMD 命令による並列実行

通常の焼きなまし法関数の SA に対し SA_PAR のような，並列実行を行うメタヒューリスティクス関数に複数の初期解を与えることで，明示的な SIMD 命令の手書きの必要なく，並列実行が行われるようにコンパイルする．

6　評価

4 種類のメタヒューリスティクスを適用するプログラムを C++ と OpTwoFast で記述し，そのプログラムの本質部分の行数と実行時間を比較した．巡回セールスマン問題における結果を表 1 に示す．行数の括弧内は C++ プログラム中でライブラリとして切り出された部分が占める行数である．行数は半分程度に削減され，実行時間の増加は数 % に収まっている．

また，巡回セールスマン問題に焼きなまし法を適用するプログラムについて，SIMD 命令により 8 並列で実行し，8 倍の計算を扱ったときの性能の変化を表 2 に示す．経路長は 3% 程度短縮され，実行時間は 2 から 3 倍の範囲に収まっている．

7　関連研究

ParadisEO[3] は，アルゴリズムを要素に分解することでメタヒューリスティクスの設計を支援する C++ フレームワークである．高速化技法を駆使して性能を高める点については改善の余地があり，ある問題を少しでも効率良く解くという目的においては採用し難い．特に，評価値やハッシュ値を自動的に差分計算することによって，高速なアルゴリズムを実装する労力を削減するという試みは行われていない．

表 2　SIMD 命令による並列化の性能

頂点数	20	40	60	80	100
経路長	97.7%	96.6%	96.4%	96.8%	96.5%
実行時間	228.3%	234.3%	243.1%	259.0%	279.2%

ispc[4] は，SIMD 並列プログラムを簡単に書くことができるプログラミング言語である．ビットマスクによってインスタンスごとに個別の制御フローを提供することを実現している．この制御フローを実現する基礎技術を，本研究で SIMD 命令による並列実行を実現するにあたって参考とした．また，ビットマスクが全て真や偽である場合を想定した最適化等により高速化をしている．本研究においてはそれらを実装していないため，性能向上の余地がある．

8　おわりに

各種メタヒューリスティクスの抽象化について検討し，多くの共通項が見出され，最小限の実装の変更によりアルゴリズムを変更することが可能となった．そして，性能を追求しながらも煩雑でないプログラムでメタヒューリスティクスの実装を可能とするドメイン特化言語である OpTwoFast を設計し実装した．複数の数理最適化問題に各種メタヒューリスティクスを適用するプログラムを作成し，C++ によるものと比較して，その有用性を確認した．

今後の課題として，並列実行には最適化の余地が多く残されているため，それらを実装することで性能のさらなる向上が期待される．また，遺伝的アルゴリズムや粒子群最適化など他のメタヒューリスティクスについても抽象化を検討し実装することで，応用範囲が広がると考えられる．

参考文献

[1] Sugie, Y. e. a.: Graph Minors from Simulated Annealing for Annealing Machines with Sparse Connectivity, *Theory and Practice of Natural Computing*, (2018), pp. 111–123.

[2] Rutenbar, R.: Simulated annealing algorithms: An overview, *IEEE Circuits and Devices Magazine*, Vol. 5, No. 1(1989), pp. 19–26.

[3] Johann Dreo, Arnaud Liefooghe, Sébastien Verel, Marc Schoenauer, Juan J. Merelo, Alexandre Quemy, Benjamin Bouvier, Jan Gmys: Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo, *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, (2021), pp. 1522–1530.

[4] Matt Pharr, William R. Mark: ispc: A SPMD compiler for high-performance CPU programming, *IEEE Innovative Parallel Computing*, (2012), pp. 1–13.