プログラミングコンテストの経験と図形の充填問題の研究

今道　貴司

本稿では，アルゴリズムの実装を競うプログラミングコンテストについて紹介するとともに，プログラミングコンテストに参加した経験が研究にどのような影響を与えたかを筆者の経験を通じて述べる．事例として筆者が行った図形の充填問題の研究を取り上げる．

キーワード：プログラミングコンテスト，充填問題，組合せ最適化

1. はじめに

昨今さまざまなプログラミングコンテストが開催されるようになってきた．読者の周辺でも何かしらのプログラミングコンテストに参加している学生や知り合いがいるのではないかと思う．プログラミングコンテストの種類は大きく分けると，与えられた課題のアプリケーションを実装するものと，与えられた条件を満たすアルゴリズムを実装するものがある．このうちアルゴリズムを実装するタイプのプログラミングコンテストで必要とされる技術は，オペレーションズ・リサーチを含めて計算機科学の研究に直接的・間接的に役に立つ．またプログラミングコンテストで得られる経験は研究者としてのキャリアにも影響を与えうるものである．

筆者は中学生の頃からプログラミングコンテストに参加することでプログラミングやアルゴリズムに対する興味を持つようになり，現在は IBM の研究所で天然資源産業における最適化の研究を行っている．その過程でプログラミングコンテストの経験は個人的に良い影響を与えてくれた．筆者のプログラミングコンテストにおける経験を紹介するとともに，研究における影響も考察する．特に筆者が以前取り組んでいた図形の充填問題でプログラミングコンテストの経験が役に立った事例を紹介する．

2. プログラミングコンテストの紹介

現在数多くのプログラミングコンテストが実施されており，筆者が知っているだけでも次のようなコンテストが開催されている：Supercomputing Contest，ACM 国際大学対抗プログラミングコンテスト，TopCoder，Google Code Jam，Microsoft Imagine Cup，Facebook Hacker Cup，ICFP Programming Contest，情報オリンピック，全国高等専門学校プログラミングコンテスト．本稿では Supercomputing Contest と ACM 国際大学対抗プログラミングコンテストを紹介する．

2.1 Supercomputing Contest (SuperCon)

Supercomputing Contest (SuperCon) は東京工業大学と大阪大学が主催する高校生と高専生を対象にしたプログラミングコンテストである．スーパーコンピュータを用いて課題に取り組むというのが大きな特徴である．参加者は最大 3 名のチームで参加し，1 週間弱の期間で与えられた課題を解くプログラムをスーパーコンピュータ上で実装する．

筆者は 1997 年と 1998 年にスーパーコンピュータに憧れて参加して，初めて Cray のスーパーコンピュータ上で並列化したプログラムを実行した際の性能に度肝を抜かれたことを覚えている．またこの大会で初めて大学という場所に足を踏み入れ，研究者の先生たちに会い，筆者が研究の世界の一端に触れることができた貴重な機会だった．

1997 年の課題は三角形のビリヤード枠が与えられたときに球の軌跡がサイクルとなるような発射位置と方向を列挙する問題で，1998 年の課題は長方形の充填問題だった．長方形の充填問題については 4 節で詳しく紹介する．

2.2 ACM 国際大学対抗プログラミングコンテスト (ACM-ICPC)

ACM-ICPC は ACM が後援する世界最大規模のプログラミングコンテストである．選手は大学内で 3 名

いまみち　たかし
IBM Research – Brazil
Av. Pasteur 138/146, Botafogo, Rio de Janeiro,
CEP: 22290–240, Brazil

1 組のチームを作って参加し，5 時間程度の制限時間内に 10 個程度の課題を解くプログラムの実装を行う．より多くの課題をより早く解くことが勝負の対象となる．コンテストの最中に使える PC が 1 台しかないので，3 名同時にプログラムを書くことができないことが特徴である．そのためチーム内での作業分担や連携が重要な戦略となる．ルールの詳細は大会のホームページを参照してほしい[1,2]．

日本ではインターネット上で行われる国内予選が最初に行われる．その後日本国内で開催される地区大会があり，これを通過したチームが世界大会に参加することができる．今年の地区大会は 11 月に東京で開催される[3]．直近の世界大会は今年の 6 月にロシアのエカテリンブルグで開催され，東京大学のチームが銀メダルを獲得した[4]．ちなみに ACM-ICPC のレポート[5]によると，この世界大会に至るまでのさまざまな予選に 94 カ国，2,286 大学から 32,043 名の選手が参加したとのことである．

ACM-ICPC で出題される課題は多岐にわたるため，全探索，動的計画法，数論，計算幾何学，グラフアルゴリズムなどの，広範な知識と実装の経験が必要となる．問題例として 2012 年の国内予選の E 問題を引用する．

> 2012 年国内予選 問題 E：鎖中経路[6]
>
> 平面上の複数の円環から構成される鎖がある．最初（最後）の円は，次（一つ前）の円だけと交差し，中間の各円は隣の二つの円だけと交差する．
>
> あなたの仕事は，以下の条件を満たす最短経路を見つけることである．
>
> 経路は，最初と最後の円の中心をつなぐ．経路は鎖中にある．すなわち，経路上のすべての点は，少なくとも一つの円の内側もしくは円周上にある．図 1 は，そのような鎖の例と対応する最短経路を示したものである．

この課題は，鎖の円同士の交点を求めたのちに，始点となる円の中心，円の交点，終点となる円の終点を結ぶ線分の中から円の鎖の中を通るものだけを用いてグラフを作り，そのグラフ上で最短路を計算することで解くことができる．この例では幾何学とグラフアルゴリズムの知識を組合せる必要がある．ACM-ICPC の過去問は公式ページにまとめられている[7]．

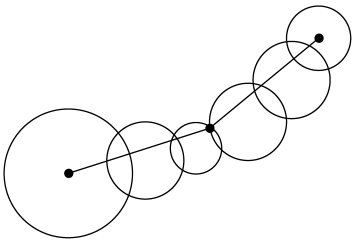

図 1　鎖の例と対応する最短路

数あるプログラミングコンテストの中で筆者にとって ACM-ICPC が最も関わりのあるコンテストである．大学生のときに 2000 年から 4 回選手として地区大会に参加した．最も成績が良かったときで地区大会 6 位だったので，残念ながら世界大会を選手として経験することはできなかった．しかし，その後コーチとして参加したチームが 2005 年と 2006 年に世界大会に出場して，筆者も世界大会を経験することができた．また，その後も OB として 2007 年に東京で開催された世界大会のボランティアスタッフとして参加した．

3. プログラミングコンテストの経験が研究活動に与える影響

プログラミングコンテストで選手は制限時間内に与えられた課題のアルゴリズムを考え，プログラムを実装し，必要に応じてデバッグを行う．制限時間は通常は長くて数時間というものなので，研究で考えるスパンよりは短く，課題もそれに応じた難易度となっている．そのためプログラミングコンテストで役立つ技術が必ずしも研究で直接役立つわけではないことは承知したうえでメリットを考察した．

3.1 基本的なアルゴリズム，データ構造の知識およびプログラミング技術の習得

プログラミングコンテストの課題のアルゴリズムは動的計画法，最短経路探索などの基本的なアルゴリズムを組合せて構成する．そのため練習では，基本的なアルゴリズムやデータ構造の仕組みを理解したうえで実装できるようになってから，さまざまな課題に取り組んで基本的なアルゴリズムの組合せる方法に慣れるようにする．課題には実行時間やメモリサイズに制限

[1] http://icpc.baylor.edu/
[2] http://icpc.iisf.or.jp/
[3] http://icpc.iisf.or.jp/2014-waseda/
[4] http://icpc.iisf.or.jp/2014/06/world-final/
[5] http://icpc.baylor.edu/download/worldfinals/pdf/Factsheet.pdf
[6] http://www.cs.titech.ac.jp/icpc2012/icpc2012-mondai/icpc2012/contest/E_ja.html
[7] http://icpc.iisf.or.jp

があるので，どんなアルゴリズムを用いても良いわけではなく，計算量も考慮に入れてアルゴリズムを設計する必要があり，練習を通じて各種アルゴリズムの性質を覚えることができる．また，入力時間を短縮するために，より簡潔にプログラムを書くための工夫も行われる．プログラミングコンテストの教科書 [1] があるので，どのような練習が行われるかの参考になる．

また，エディタ，コンパイラ，デバッガなどのプログラムの実装に必要な基本的なツールは自在に扱うことができるように練習が行われる．これらの練習を通じて得られる技術は，実装が伴う種類の研究では必須となる技術であり，プログラミングコンテストの練習を通じて事前に訓練をしている学生はより研究に取り組みやすいはずである．

3.2 チームワークの向上

プログラミングコンテストの中には，ACM-ICPC のように複数人でチームで課題に取り組む場合がある．特に ACM-ICPC の場合ではチームで使える PC が 1 台しかないという性質上，メンバーで効率的に役割分担をする必要がある．英語の問題文を読解する係，アルゴリズムを考える係，プログラムを実装する係，デバッグをする係などがある．そして，担当間で意思疎通を正確かつ効率的に行うことが大切である．このような正確な意思疎通は研究においても指導教員や同僚との間で研究の議論をする際に必要な技術である．

3.3 交流の広がり

プログラミングコンテストに参加することでさまざまな人と出会えることは貴重な経験である．例えば ACM-ICPC の場合では大学内からインターネットで参加する国内予選，日本国内の会場に選手が集まる地区大会，そして世界大会がある．これらの会場では同じ目標を持って切磋琢磨しているライバルたちと実際に会うことができ，この場での情報交換は刺激的である．

またライバルの選手以外にも，主催者やスポンサーとの交流もある．企業が主催するプログラミングコンテストの場合は一般的に，企業の宣伝をするためのイベントも催されて，学生たちは企業の担当者から話を聞くことができる．ACM-ICPC の場合では，地区大会や世界大会の懇親会にはスポンサー企業の担当者が出席する．このような交流が研究者としてのキャリアに影響することがある．

筆者の場合は，ACM-ICPC に参加した際に IBM 東京基礎研究所の話を聞いて興味を持ったのがきっかけで，研究所の見学をさせてもらい，その後博士課程の際にはインターンとしてお世話になって企業の研究所での活動を体験することができた．そして博士課程修了後は IBM 東京基礎研究所でポスドクをした後に就職している．また個人的に経験した最も大きなイベントとしては，2005 年の ACM-ICPC 上海世界大会後に，Google が世界大会参加チームを Mountain View の本社に招待したイベントがある．Google の職場の様子や最新の技術について知ることができただけでなく，創業者の一人のセルゲイ・ブリン氏の講演もあり豪華だった．

4. 長方形の充填問題

前節ではプログラミングコンテストでの経験が研究に与える影響を考察したが，主に間接的なものだった．しかし筆者は一度，自分が関係する研究でプログラミングコンテストの経験が直接的に関係した事例があったので，本節で紹介する．

ここでは長方形の充填問題を取り上げる．入力として与えられる長方形の集合を長方形の容器の中に重なりなく配置する問題である．目的関数として，容器の数を最小化するビンパッキング問題や，容器の長さを最小化するストリップパッキング問題などの種類がある．ちなみに，長方形の充填問題を含む各種切出し・充填問題の分類は Wäscher らの論文が詳しい [2].

4.1 定式化

ここでは長方形のストリップパッキング問題の定式化を示す．入力として長方形の容器の幅 W と容器の中に配置する長方形の集合 $I = \{1, \ldots, n\}$ が与えられる．長方形 i の幅と高さはそれぞれ w_i，h_i とする．決定変数は容器の高さ H と各長方形の左下の角の座標 (x_i, y_i) である．この時長方形の回転を許さない場合のストリップパッキング問題は以下のように定式化できる．

$$
\begin{aligned}
\text{minimize} \quad & H \\
\text{subject to} \quad & x_i + w_i \le W, \quad i \in I, \\
& y_i + h_i \le H, \quad i \in I, \\
& x_i + w_i \le x_j \text{ or } x_j + w_j \le x_i \text{ or} \\
& \quad y_i + h_i \le y_j \text{ or } y_j + h_j \le y_i \\
& \quad i, j \in I, i \ne j, \\
& H \ge 0, \\
& x_i, y_i \ge 0, \quad i \in I.
\end{aligned}
$$

4.2 長方形の充填問題の研究に取り組むまでの経緯

筆者が長方形の充填問題に初めて取り組んだのは高

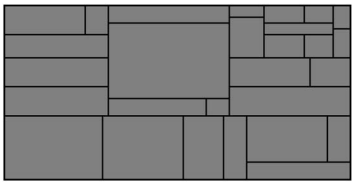

図 2　長方形を隙間なく配置する例

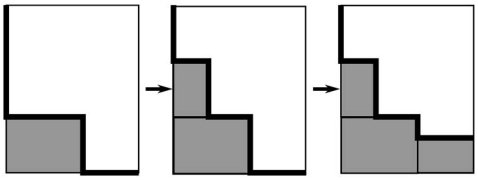

図 3　階段上に長方形を 1 枚ずつ配置する例

校生のときだった．1998 年の 8 月の東工大の SuperCon の本選[8]が長方形の充填問題の一種だった．入力の容器の大きさが固定で，入力の長方形で容器を隙間なく充填することができるようになっており，長方形の配置を求めるのが課題だった．図 2 はそのような配置の例である．

このとき優勝した水戸第一高校の泉氏の解法は，長方形を 1 枚ずつ順次配置していく際に，配置済みの長方形の領域が階段状になるように階段の角へ配置していき，無駄なスペースが発生する場合はバックトラックをするというものであった（図 3）．またヒューリスティクスとして，縦横比の小さい長方形は 90 度回転させないことや，配置済み長方形による階段の段数が 3 段以上になる場合もバックトラックする工夫なども加えられていた．詳しくは解説記事にて紹介されている [3]．筆者はこの大会に選手として参加して，表彰式で泉氏の発表を聞いた際にそのアルゴリズムが簡潔ながら非常に効果的であることに感銘を覚えた．

時は変わって，2004 年に筆者が研究室で後輩の研究の進捗報告会を聞いていた際に，ある学生のテーマが長方形のストリップパッキング問題であったことから，筆者は 1998 年の泉氏の SuperCon 解法を思い出して，ストリップパッキング問題でも有効ではないかと思いたった．実際に実装して試してみたところ有効であることが確認できたので，その学生は長方形を階段上に配置する解法を基に，分枝限定法による長方形のストリップパッキング問題に対する厳密解法の研究を行った [4]．

その研究に関する文献調査を筆者たちが行っていた際に面白い点を発見した．Martello ら [5] が 2000 年に 3 次元のビンパッキング問題に対する厳密解法を発表しているが，その中で 2 次元と 3 次元の場合に対して階段状に長方形や直方体を配置する手法を提案していた．このアイデアは 1998 年 8 月の SuperCon の泉氏のアイデアと同じであった．Martello らの論文は投稿が 1997 年 6 月で採択が 1998 年 10 月となっていて，Martello らのほうが着想は早いものの，ほぼ同時期に同じアイデアを高校生が考えついていたことは興味深い．

4.3　無駄な分枝操作を減らす工夫

筆者らはその後も長方形のストリップパッキング問題に対する厳密解法を研究しており，その中で階段状に長方形を配置するアルゴリズムの分枝操作の無駄を減らす手法を提案している [6]．本節でその概要を紹介する．アルゴリズムの全体の流れは，容器の高さを固定したときに長方形が配置可能かどうかを判定する部分問題を用いて，目的関数の容器の高さは二分探索で求めるというものである．容器の高さを固定した際の長方形の配置可能性の判定は分枝限定法によるアルゴリズムを用いる．分枝操作として階段上に長方形を 1 枚ずつ配置する手法を用いる．

分枝操作の際に同一の長方形の配置を重複して探索する例を図 4 を用いて説明する．図 4 の長方形 a と b が配置される前の状態を考えた場合に，長方形 a を階段の角 1 に配置した次の分枝操作の一つとして長方形 b を階段の角 3 に配置した状況も探索することになる．その後バックトラックにより長方形 a が配置される前の状態に戻って，今度は長方形 b を角 3 に配置した次の分枝操作で長方形 a を階段の角 1 に配置する場合が再び現れることになる．このように分枝限定法の探索木の中で複数の親ノードから同一の子ノードへの分枝が発生しうる．

この状況を改善するために，筆者らは探索木の各ノードに対して複数存在しうる親ノードの中から唯一の親ノードを選ぶルールを導入した．そして長方形を 1 枚階段の角に配置して新しい配置を作って分枝操作をする際には，その配置の親ノードに対応する配置と現在の配置が一致しているかを判断することで，決められた親ノードから分枝した子ノードのみ探索を続けるようにする．これより図 4 の配置も探索中に一度しか現れなくなる．

具体的な親ノードを決定するルールとしては，長方形の配置の中から一番上にある除去可能な長方形を 1 枚除去した配置を親の配置とするものを提案した．長方

[8] http://www.gsic.titech.ac.jp/supercon/supcon98-j/honsen-kadai.html

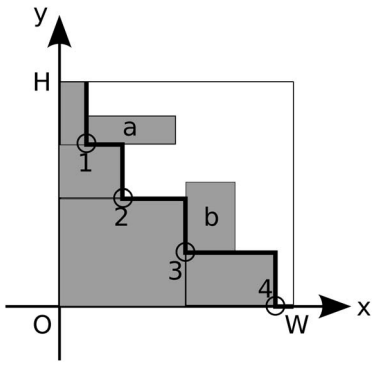

図 4　配置済みの長方形の階段の角に新たに長方形 a と b を配置する例

形が除去可能とは，その長方形の上方向と右方向の両方向に平行移動させた場合に衝突する長方形が配置されていない状態であることと定義する．実装上は，長方形を配置していく際に最後に配置した長方形の底辺の y 座標 y' を更新しておき，新しく長方形 i を階段の角 (x, y) に配置する場合には，

$$y + h_i > y'$$

をみたせば分枝操作を行う．もし $y + h_i \leq y'$ の場合は，最後の配置した長方形のほうが長方形 i よりも上にあってかつ除去可能となるため分枝しない．図 4 の場合では長方形 a を階段の角 1 に配置した直後に長方形 b を階段の角 3 に配置する分枝操作を行うことがなくなる．

5.　非凸多角形の充填問題

本節では筆者が多角形の充填問題の研究を行った際の経験を紹介する．

5.1　定式化

非凸多角形のストリップパッキング問題は，容器内に多角形を配置する点で前節の長方形のケースとは異なる．入力として，長方形の容器の幅 W と多角形の集合 $I = \{P_1, \ldots, P_n\}$ が与えられる．多角形は非凸でも構わない．ただし，各多角形 P_i は辺と内部の点の集合として，また原点を各多角形の参照点とする．決定変数は容器の高さ H と各多角形 P_i の参照点の位置 (x_i, y_i) である．多角形 P_i の参照点を (x_i, y_i) へ移動した多角形を Minkowski 和を用いて

$$P_i \oplus (x_i, y_i) = \{\boldsymbol{p} + (x_i, y_i) \mid \boldsymbol{p} \in P_i\}$$

で表す．また，容器の辺と内部の点の集合を $C(W, H)$，集合 S の内部を $\mathrm{int}(S)$ で表す．非凸多角形のストリップパッキング問題は以下のように定式化できる．

$$\begin{aligned}
\text{minimize} \quad & H \\
\text{subject to} \quad & P_i \oplus (x_i, y_i) \subseteq C(W, H), \quad P_i \in I, \\
& \mathrm{int}(P_i \oplus (x_i, y_i)) \cap (P_j \oplus (x_j, y_j)) = \emptyset, \\
& \quad P_i, P_j \in I, \quad i \neq j, \\
& H \geq 0, \\
& x_i, y_i \geq 0, \quad i \in I.
\end{aligned}$$

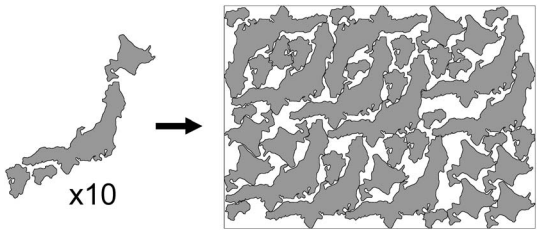

図 5　北海道，本州，四国，九州を 10 枚ずつ長方形の容器に配置した例

非凸多角形の配置例として，北海道，本州，四国，九州を 10 枚ずつ 180 度の回転を許して長方形の容器内に配置した結果を図 5 に載せる．

5.2　本問題に取り組んだ際の経験

筆者が大学 4 年生でこの問題の研究に取りかかった際に，当時研究室ではプログラミング言語として C か C++の選択肢があったが，プログラミングコンテストの経験を通じて C++に慣れていたことが役に立ったと感じている．

C++は C に比べて豊富な標準ライブラリが含まれている．筆者はプログラミングコンテストの練習を通じて，それらのデータ構造を利用したアルゴリズムの実装に慣れていて，また多角形内の点の存在判定や多角形の凸分解などの，計算幾何学の基本的なアルゴリズムの実装の経験があった．研究で非凸多角形のアルゴリズムを実装する際には，その経験を活かすことができた．

さらに，C++の標準ライブラリの実装はすでに最適化されているので，C で自ら新たにデータ構造などを実装するよりも有利である．非凸多角形のストリップパッキング問題はベンチマーク問題例が公開されており，論文を書く際は計算実験の結果を既存手法と比較する必要がある．そのため，生成されるプログラムの速度が重要だった．

ただし，これは筆者が大学で研究を行っていた 2000 年代半ばの事情で，現在は異なる選択肢もありうる．

5.3　提案した手法の概要

筆者らは非凸多角形のストリップパッキング問題に

対してメタ戦略に基づく手法を提案した [7]．全体の流れとしては，容器の高さを少しずつ小さくしていき，そのたびに大きさが固定された容器の中での多角形同士の衝突と多角形の容器からの突出を除去する部分問題を解くというものである．この部分問題を解くために反復局所探索法を基にしたアルゴリズムを構成した．すべての図形同士の衝突と，図形の容器からの突出のペナルティの総和を最小化する制約なしの非線形最適化問題を図形の貫通深度を用いて導入した．目的関数は以下のとおりである．

$$f(\boldsymbol{x})=\sum_{i=1}^{n}\sum_{j=i+1}^{n}\delta(P_i\oplus(x_i,y_i),P_j\oplus(x_j,y_j))^2 + \sum_{i=1}^{n}\delta(P_i\oplus(x_i,y_i),\overline{C(W,H)})^2. \quad (1)$$

なお，$\overline{C}$ は集合 C の補集合として，$\boldsymbol{x}=(x_1,y_1,x_2,y_2,\ldots,x_n,y_n)$ とする．貫通深度は衝突している図形を分離するのに必要な平行移動の距離で，以下のように定義される．

$$\delta(P,Q)=\min\{\|\boldsymbol{z}\| \mid \mathrm{int}(P)\cap(Q\oplus\boldsymbol{z})=\emptyset,\ \boldsymbol{z}\in\mathcal{R}^2\}.$$

$f(\boldsymbol{x})$ が微分可能なので，準ニュートン法を適用することで局所最適解を高速に得ることができる．さらに，図形同士をランダムに交換して局所最適解に摂動を加える操作を組合せることで反復局所探索法のアルゴリズムを構成した．

5.4 No-fit polygon (NFP)

本アルゴリズムの実装上のポイントとなるのが，多角形の配置に対するペナルティ関数 $f(\boldsymbol{x})$ とその勾配の計算である．

多角形の充填問題では多角形同士の衝突判定が基本的な操作として重要だが，元の図形のまま衝突判定を行うのは難しい．そこで多角形が回転しない場合や，回転する角度の選択肢が限られている場合は，*No-fit polygon* (*NFP*) と言われる図形を各多角形の組ごとに前処理で求めておくことで，衝突判定を高速に行う手法が一般的に用いられている．図形 P の図形 Q に対する NFP は，Minkowski 和を用いて以下のように定義される．

$$\mathrm{NFP}(P,Q)=\mathrm{int}(P)\oplus(-\mathrm{int}(Q)) =\{\boldsymbol{p}-\boldsymbol{q} \mid \boldsymbol{p}\in\mathrm{int}(P),\boldsymbol{q}\in\mathrm{int}(Q)\}.$$

直観的には，P の参照点を原点に置いて，P に沿って Q を平行移動させた時の Q の参照点の軌跡が $\mathrm{NFP}(P,Q)$ になる（図 6）．

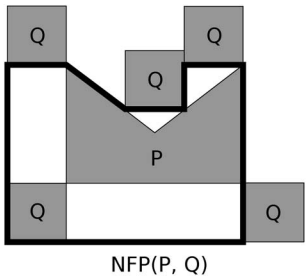

図 6　多角形 P と Q の No-fit polygon を黒の太線が表す

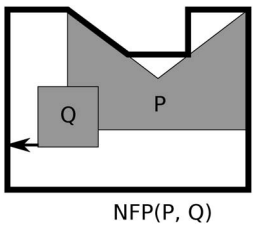

図 7　多角形 P と Q の貫通深度を矢印の長さが表す

$\mathrm{NFP}(P,Q)$ を用いると $P\oplus\boldsymbol{x}$ と $Q\oplus\boldsymbol{y}$ の衝突判定は参照点の位置の差のベクトル $\boldsymbol{y}-\boldsymbol{x}$ が $\mathrm{NFP}(P,Q)$ の中に含まれるかを判定するのと等価になる．すなわち NFP の多角形の中に点が含まれるかどうかを判定することで，元の多角形同士が衝突しているかを判定することができる（図 7）．

また，多角形の配置のペナルティ関数で用いる貫通深度も NFP を用いることで容易に計算することができる．多角形の組が衝突している場合は，その参照点の差のベクトル $\boldsymbol{y}-\boldsymbol{x}$ は NFP の中に含まれていて，そのベクトルから最も近い NFP の辺までの距離が貫通深度と等しくなる（図 7 の矢印の長さ）．多角形の組の衝突ペナルティの勾配も，$\boldsymbol{y}-\boldsymbol{x}$ から最寄りの NFP の辺までのベクトルから容易に求めることができる．

以上よりすべての多角形の組に対して NFP を前処理で求めることで，多角形の配置のペナルティ関数 (1) とその勾配を効率的に計算することができる．

6. 最後に

ここまで，プログラミングコンテストの紹介および研究に与える効果を考察するとともに，筆者が充填問題の研究を行う際の体験を事例として紹介してきた．プログラミングコンテストの経験は，特に実装の伴う研究を行う際には有用で，またプログラミングが好きな面白い人たちと会って交流を広めたり情報交換ができる貴重な機会である．筆者はプログラミングコンテ

ストに興味のある学生には積極的に参加してほしいと思うとともに，教育機関においても，学生のプログラミングコンテストへの参加が一層推奨されるようになることを望んでいる．

参考文献

[1] 秋葉拓哉，岩田陽一，北川宜稔，『プログラミングコンテストチャレンジブック』，第 2 版，マイナビ，2012.

[2] G. Wäscher, H. Haußner and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, **183**, 1109–1130, 2007.

[3] 松田裕幸，第 4 回東工大スーパーコンピュータコンテスト，数学セミナー，447, 1998 年 12 月.

[4] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura and H. Nagamochi, "Exact algorithms for the two-dimensional strip packing problem with and without rotations," *European Journal of Operational Research*, **198**, 73–83, 2009.

[5] S. Martello, D. Pisinger and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, **48**, 256–267, 2000.

[6] Y. Arahori, T. Imamichi and H. Nagamochi, "An exact strip packing algorithm based on canonical forms," *Computers & Operations Research*, **39**, 2991–3011, 2012.

[7] T. Imamichi, M. Yagiura and H. Nagamochi, "An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem," *Discrete Optimization*, **6**, 345–361, 2009.

