問題解決エンジンとしてのメタヒューリスティクス・アルゴリズム

茨木　俊秀
関西学院大学理工学部
情報科学科

669-1337 三田市学園 2-1
ibaraki@ksc.kwansei.ac.jp

Abstract

組合せ最適化問題はさまざまな応用場面に現れるが, そのほとんどが NP 困難である. NP 困難であっても適切な近似アルゴリズムによって対応できる場合が多いとはいえ, アルゴリズムの開発は容易ではない. この隘路を解決するため, いくつかの標準問題を定義し, それらに対する問題解決エンジンを準備するというアプローチを提案する. この目的にはメタヒューリスティクスが適当であることを述べ, これまで開発した標準問題とそれらのエンジンを紹介する. 具体的例として, 資源制約スケジューリング問題についてやや詳しく述べる.

Keywords: 組合せ最適化, 問題解決エンジン, 標準問題, 汎用アルゴリズム, メタヒューリスティクス, 資源制約スケジューリング問題.

1　はじめに

現代社会において解決を求められている問題の多くは数理的に定式化される. それらは規模の拡大と複雑化の一途をたどっている. コンピュータの驚異的な進歩によって, 大幅な計算能力の増大が実現されたといっても, それだけで対応できるわけではなく, 個々の問題に対して, 高度な数学的手法の利用と, 大規模なソフトウエアの開発が必要である. そのため, 個別の問題ごとに十分なマンパワーを投入することは現実には不可能であろう. すなわち, 問題のモデル化とアルゴリズムの開発において, 新しいアプローチが必要になる.

一つの考え方として, 汎用で効果的なソフトウエアである「問題解決エンジン」を開発して, 多様な問題を一括して処理させるというアプローチがある. もともとコンピュータはあらゆる問題を解決してくれる魔法の箱だという素朴な認識があった. コンピュータにできることにはもちろん限界があるが, 分野を限定すれば, このような問題解決エンジンの構築が可能であるかも分からない.

たとえば, 人工知能の分野では, 1950 年代, GPS (general problem solver) といった野心的な名称の下に, 数学のあらゆる問題を論理的な枠組みで解決するという目標が掲げられた. 究極の姿は, 問題の定義を論理的な言葉で正確に入力すれば, 解決策は「推論エンジン」が自動的に導くというものである. この流れは, その後の論理プログラミング, エキスパートシステム, 制約プログラミングなどに継続しており, 一定の実績をあげている. 推論を助ける規則, 対象領域において知られている知識, さらに学習能力などをさらに取り込んで, 効率化が図られているのはよく知られているところである.

数理計画の諸アルゴリズムも, 最適化問題に対する問題解決エンジンの構築を目指しているといえる. 線形計画法の大きな成功 (シンプレックス法や内点法) に牽引され, 非線形計画や整数計画の分野でも着実な進歩が続いている. 商用パッケージも種々開発され, 実用的にも大きな成功をあげている.

ここでは, 組合せ最適化の分野での問題解決エンジンについて考察する. この種の問題は, 生産, 通信, 交通など, 毎日の生活に密接した問題として数多く現れる. 整数計画法はこれら

の問題に適用される有力な手法であるが, 後述のように, すべてを処理するには無理がある. 複数個の標準問題を定義し, それらに対する問題解決エンジンを構築することによって, はじめて総合的な対応が可能になると考えている. さらに, 「問題解決エンジン群」の実現にあたっては, 計算量 (時間量と領域量) の壁をいかに克服するかを明らかにしなければならない. 以下, この点に関する我々の考え方と, これまでの実績を紹介する.

2 問題解決エンジン群による汎用アプローチ

2.1 NP 困難性と近似アルゴリズム

個々の問題が持つ計算の複雑さを明らかにするための理論は, 1950〜60 年代に基礎が固まり, 1970 年代になって NP 完全性と NP 困難性の概念が導入されると, 飛躍的に発展した. よく知られているように, P≠NP 問題はこの分野の最大の未解決問題として残されているものの, NP 完全性の概念を利用して, 計算量の意味で実用的に扱える問題 (具体的には, 多項式オーダー時間で解ける問題) とそうでない困難な問題との分類が詳細になされるようになった.

NP というのは, 次のような性質をもつ問題のクラスである. すなわち, NP の問題は, 与えられた条件 (たとえば, 与えられたグラフのすべての点を一度ずつ通るハミルトン閉路という条件, など) をみたす解が存在するかどうかを問う形に書かれる. より詳しく述べると, 解 (たとえば, グラフの n 点の順列) のすべての場合を列挙できて (その数は多項式オーダーでなくてもよい), それぞれの解に対する条件 (その順列が, 与えられたグラフの閉路になっているかどうか) の判定が多項式時間で可能であり, 一つでも条件をみたす解があれば, 答えとしてイエスを出力する, というタイプの問題である. 現実に現れる組合せ問題のほとんどはクラス NP に属していると考えられるので, 以下の議論はこのクラス (あるいはその最適化版) を念頭において進める.

ある問題 A が NP 完全であるというのは, 詳しい説明は略すが, クラス NP の中で, どの問題と比べても A がそれ以上に困難であると示せることを意味する. 換言すれば, クラス NP の中で最も困難な問題だということである. クラス NP は判定問題だけを扱うので, その中に最適化問題は含まれないが, NP 完全問題以上に困難であることを NP 困難と呼び, 最適化問題も扱えるようにする. 以下あまり明確に区別せずに, 誤解のない限り, NP 困難という言葉を主に使う.

1972 年, S. Cook は問題 SAT (satisfiability, 充足可能性問題) が NP 完全であることを示した. その後, SAT の結果から出発して, 多くの問題が NP 完全あるいは NP 困難であることが証明された. たとえば, 上で例として用いたハミルトン閉路の問題, 1 次不等式系を満たす整数解の存在を問う整数計画問題 IP など, よく知られた組合せ最適化問題がその中に含まれる.

NP 困難問題の代表例として IP を用いると, その NP 困難性はつぎの二つの意味を持つ.

1. クラス NP のすべての問題は IP に帰着できる.
2. IP を多項式時間で解くことはできない (と予想される).

性質 1 は通常あまり強調されないが, 本稿の目的には大変重要である. つまり, IP を効率よく解くことができれば, 図 1 のように, IP を通してクラス NP の問題をすべて扱うことができるということを意味している. これこそ「問題解決エンジン」でなくて何であろう. しかし残念ながら, このアプローチには, 性質 2 にあるように, 計算量の壁が立ちはだかっているのである.

しかし, 問題解決を迫られている現実の場では, 厳密解がいつも求められているわけではないであろう. 最適解でなくても, 最適値からの違いが少なければ, 近似解として実用的に十分使えるのではないか. つまり, IP 問題に対して, 良質の解を高速に求めることができれば,

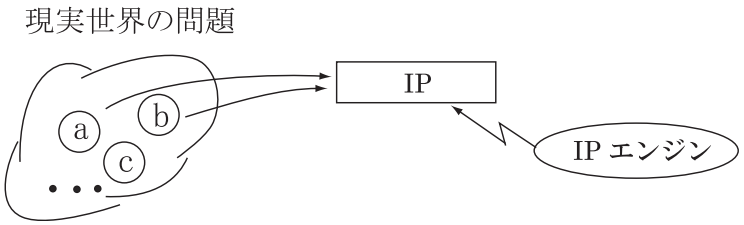

Figure 1: 問題解決エンジン

事実上「問題解決エンジン」の役割を果たすことになる. 実際, IP ソルバは, 多くの現実問題に適用されているが, 大抵の場合, 近似ソルバとして利用されている.

困難な組合せ問題を近似的に解くアルゴリズムの研究は, 現在最もホットな分野の一つで, 理論的にも, 保証された近似比をもつアルゴリズム, 確率的な性能評価, 確率アルゴリズム, オンラインアルゴリズムなど, さまざまな枠組みで研究が進んでいる.

しかし, このアプローチにも本質的な困難が隠れている. まず, クラス NP の任意の問題が, IP に帰着できるといっても, これはある理論的な枠組みで可能という意味で, 帰着の際に問題の規模がずっと大きくなってしまう場合がある. 本来 n 個の変数をもつ問題を IP に帰着するとき, 必要な整数変数の個数が n^2 あるいは n^3 のような数に増えるのであれば, $n = 1000, 10000$ といった問題例には適用できないであろう. つぎに, 帰着が可能であったとしても, IP に対して求められた良質の近似解が, 帰着の元となった問題の近似解として良質とは限らないという問題がある. これは, 帰着の際の変形が, 解を評価するための目的関数をも歪めてしまうことに起因している. 厳密な解は保存しても (それが帰着の意味であるが), 厳密解からの近似解のずれは保存しない場合がある. 結局, 近似解で満足するとしても, ただちに「問題解決エンジン」につながることにはならない.

2.2 二つのアプローチ

以上の考察からわかることは, 1 台のエンジンですべての問題を扱うのはやはり無理があるということである. この隘路を逃れるために, 次の二つのアプローチが提唱されている.

一つは, アルゴリズムの一般的枠組みをソフトウエアとして実現しておき, その個々の構成要素を問題ごとに具体化するというアプローチである. 枠組みの具体例として, 後述するメタヒューリスティクスを考えると, その構成要素には, 近傍の定義, 近傍の探索法, 解の移動規則, 評価関数の定義と変更規則, 制約条件の定義, ペナルティの定義, 候補解の集合, 新しい初期解の生成法など, 種々あげることができる. これらの構成要素の細部およびそれらの組合せ方は, 解くべき問題が具体的に定まってはじめて決定されるが, そのための労力をできるだけ軽減することが求められる. 理想的には, 問題の記述を適当な言語で与えると, それに基づいて各構成要素が自動的に構成されるというのが望ましい. 各構成要素の記述の中に, 問題定義に基づいて細部を決定するアルゴリズムを装備しておくことによって, この種のシステムを実現できる可能性があって, どの程度の自動化を目標とするかによって種々のモデルを考えることができる. このタイプのシステムは, たとえば文献 [30] にいくつかの例を見ることができる.

もう一つのアプローチは, 汎用性のある標準問題をいくつか想定し, それぞれに対してエンジンを開発しておくというものである. 各エンジンは, 個々の問題に特化した高性能なものでなければならない. このアプローチでは, ある問題の解が必要になったとき, 自然な形で (つまり, 問題規模を爆発させることなしに) 定式化可能な標準問題をリストの中から選び, その標準問題のエンジンを用いて解を得ることになる. この様子を下図に示す. 標準問題は当

然 NP 困難であるから, 有効な近似アルゴリズムを用いる. しかし, 標準問題への定式化は自然なものであると想定しているので, この場合の標準問題の近似解は, 元の問題の近似解として利用できることが期待できる.

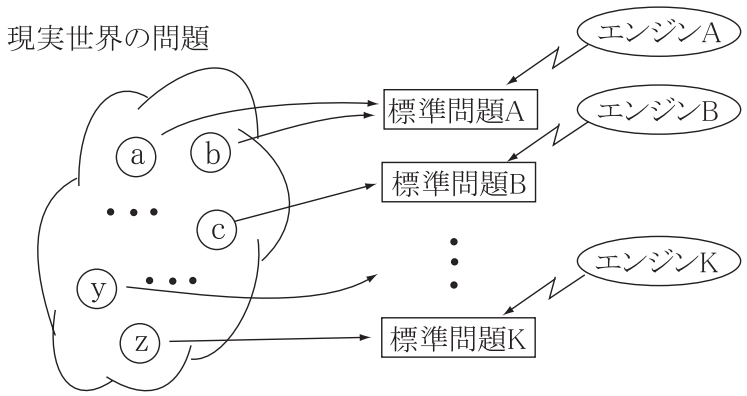

Figure 2: 標準問題によるアプローチ

上記の二つのアプローチは, それぞれ得失があるが, 現実性が高いのは後者ではないかと考えている. 以下に述べる我々の試みも, 後者の考えに沿うものである.

2.3 標準問題のリスト

標準問題のアプローチを採用するとして, つぎに議論しなければならないのは, どのような標準問題を選び, どのようなエンジンを実装するかである. もちろん, 上で言及した整数計画問題 (IP) は代表的な標準問題の一つである. IP に対してはすでにすぐれた商用のソフトウエアがいくつか開発されていて, 広く用いられている. 我々は, IP 以外の標準問題の必要性に着目し, これまで研究と開発を進めてきた. 現時点の標準問題のリストは以下のようになっている. それぞれ, 汎用性があって, しかもその問題の構造を利用すれば効率よいアルゴリズムの開発が可能であるという特性を有している.

1. 制約充足問題 (constraint satisfaction problem, CSP)
2. 資源制約スケジューリング問題 (resource constrained project scheduling problem, RCPSP)
3. 配送計画問題 (vehicle routing problem, VRP)
4. 2 次元箱詰め問題 (2-dimensional rectangle packing problem, 2PP)
5. 一般化割当問題 (genaralized assignment problem, GAP)
6. 集合被覆問題 (set covering problem, SCP)
7. 最大充足可能性問題 (maximum satisfiability problem, MAXSAT)

問題 1〜4 は, IP による定式化では変数や制約条件の数が多くなってうまく扱えないタイプの問題であり, 問題 5〜7 は特殊な IP 問題の形をしていて, 一般の IP よりさらに高速に解くことを狙って選ばれたものである.

この中で, これまで比較的多くの方に利用してもらったのは CSP と RCPSP である. CSP は IP に比べ制約条件の設定がより自由で, たとえば指定された変数集合がすべて異なる値を取ることを要求する all-different 制約なども許される. IP に比べると, 組合せ的制約を持つ問題に適しており, スポーツの対戦表や, 病院, 企業での勤務表の作成, 種々のパズルにおける解の探索などに利用されている. RCPSP については, 後でやや詳しく述べる. VRP も現

実への応用が期待される問題である. 時間枠制約も含め, より柔軟な定式化を許すように工夫してきたが, 標準問題として具体的に適用するには, まだ克服すべき課題があると考えている. 2PP については, 切断パターン問題 (cutting stock problem) への拡張, GAP については, 2 次の目的関数や施設配置問題への拡張も試みている.

現実問題への適用を考えると, 標準問題は汎用性と柔軟性をもたねばならない. 現実問題は, 大規模であるだけでなく, 適用現場の特性を反映して, 特殊な制約条件や副次的な目的関数を持つことが多い. また, すべての制約条件をみたす解が存在しない場合もあって, そのときには, 重要な制約条件はみたしつつ, 残りの制約条件については充足の度合いを高めることが求められる. これら複雑な状況に対応できるように, 上記の標準問題では, 本来の目的関数に加え, 他の副次的な目的関数や, さらに制約条件をどの程度破っているかを示すペナルティ関数を導入し, それらの重み付け和を最適化するという形によって汎用性を高めている.

3 アルゴリズムの枠組

標準問題を解くためのアルゴリズムには何が期待されるだろうか. まず, (i) 大規模な問題例も実用的に扱えるように, 時間量と領域量の両方の意味で, 効率の高いものでなければならない. それだけでなく, (ii) わずかな問題変形に対して大きく性能が劣化することのないよう, 頑健性が求められる. さらに, (iii) 利用者にとって使いやすいものでなければならない. 最後の (iii) は, 具体的には, 問題データの入力が簡単で, 見やすい形の出力が得られること, また, アルゴリズムに含まれるパラメータの調整に多大の労力が求められることがないこと, などを意味している.

このような特性をもつアルゴリズムとして, 我々は, 局所探索 (local search, LS), さらに LS を部分アルゴリズムとする, いわゆるメタヒューリスティクス (mataheuristics, メタ戦略ともいう) の枠組みに着目した.

メタヒューリスティクスは, 遺伝アルゴリズム (genetic algorithm, GA), 進化計算 (evolutionary computation, EC), アニーリング法 (simulated annealing, SA), タブー探索 (tabu search, TS), 反復局所探索 (iterated local search, ILS), 可変近傍探索 (variable neighborhood search, VNS) などを実現例として含む大きな枠組みである. その内容については, 以下に 2 節を設けて説明する. メタヒューリスティクスの研究は, 最近十数年の間に急速に進み, 国際会議も頻繁に開催され, 全般的な解説を含んだ書籍も出版されている [23, 1, 29, 25, 8, 5, 13, 31].

ところで, メタヒューリスティクスのような比較的 “重い”アルゴリズムが実用的に可能になった背景には, パソコンの驚異的な性能向上がある. 単純な局所探索だけでなく, 局所探索を何度も反復することも苦にしなくなったというのがその理由である. しかも, このような計算を, どこでも簡単に実行できるのである. これらの結果として, 本稿のような「問題解決エンジン」の構築が可能になったと言えよう.

3.1 局所探索アルゴリズム

局所探索 (LS) とは, は適当な初期解 x から始め, x の近傍 $N(x)$ 内に x の改善解 x' が存在すればそれに移動する, という操作を変化が生じなくなるまで反復するという方法である. $N(x)$ 内に改善解が存在しない場合, x を局所最適解 (locally optimal solution) という.

アルゴリズム LS
Step 1: 初期解 x を求め, $k := 1, x^{(k)} := x$ とする.
Step 2: $x^{(k)}$ の近傍 $N(x^{(k)})$ 内に $x^{(k)}$ の改善解 x' が存在すれば $x^{(k+1)} := x', k := k+1$ として Step 2 に戻る. 改善解が存在しなければ, 局所最適解として $x^{(k)}$ を出力して計算終了する.

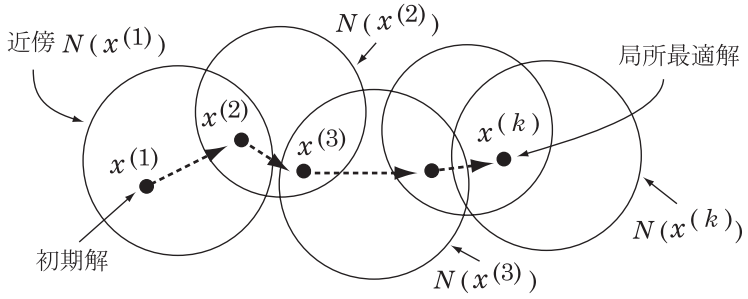

Figure 3: 局所探索

LS を実現するには, まず解の定義領域である解空間を定める必要がある. 解 x の近傍 $N(x)$ とは, 解空間において x に少しの変形を加えることによって得られる解の集合を意味する. LS の性能は, 近傍の定義によって大きく違ってくる. 問題構造を利用することで, できるだけ小さなサイズでしかも改善解を含む可能性の高い近傍をいかに構成するかがポイントである. また, Step 2 で近傍内の解をどのような順序で探索するのか, 最初に見つかった改善解にただちに移動するか, あるいは近傍内の解を全部探索した後, 最善の解に移動するかなどの細部も定めなければならない. Step 1 の初期解の定義も重要である. これについては次のメタヒューリスティクスの中で議論する. 結局, LS の性能は, これらの細部の内容と実装法 (データ構造) に依存している. 効率よいソフトウエアとして実現するには, 開発の過程で多くの工夫が必要である.

3.2 メタヒューリスティクス

メタヒューリスティクスは, LS による局所最適解の計算を内部に含み, 反復の度に初期解の生成から始めることでより柔軟性の高い枠組みを提供する. なお, 暫定解とは, その時点までの計算で得られた最良解のことである.

メタヒューリスティクスの枠組み
I (初期解生成): 初期解 x を生成する.
II (局所探索): x を (一般化された) LS によって改善する.
III (反復): 終了条件がみたされれば暫定解を出力して探索を終了する. さもなければ I に戻る.

I の初期解の生成には, その都度ランダムに生成するなど簡単な方法を用いることもあるが, それまでの計算経過に基づいた工夫がこらされているのが普通である. 過去の計算で得られた複数の良質解を記憶しておき, その中の解を何らかの形で変形するというものが多い. たとえば反復探索法では, 記憶されている最善の解に摂動を加えて初期解を生成する. GA では記憶されている複数の解に交叉という操作を加えて初期解を生成する. また, 散布探索法 (scatter search) では, 複数の参照点を利用して, それらの線形結合に基づいて初期解を生成する.

II の局所探索の部分は, 上記の標準的な LS をさらに一般化して用いることを許している. たとえば, SA では, 近傍内の解をランダムに探索し, 改善解でなくてもそちらへの移動をある確率で行う. 移動確率は温度とよばれるパラメータで調節される. TS では, 近傍内に改善解が存在しないときも, その中の最良解に必ず移動する. ただし, 解のサイクリングを避けるため, タブーリストを設けて, 最近すでに試された解への移動を禁止するなどの処置をとる. 近傍についても, 1 種類だけでなく複数の近傍を準備しておき, 計算の進行にともなって適切

な近傍を選択するという方針をとることもある. 典型的な例は, VNS にみられるように, 解の改善が得られないときには次第に近傍のサイズを大きくして, 広い範囲の探索機能を持たせるというものである.

III の終了基準も, 一定の時間が経てば終了するという簡単なものから, 計算経過に依存した複雑なものまで, 種々提案されている.

4 エンジンの実装

上記の標準問題それぞれに対し, メタヒューリスティクスのアルゴリズムを組み込んだエンジンを開発し, 多様な問題例に適用する実験を進めてきた. エンジンの開発では, 解空間をどのように定義し, その中でどのような近傍を採用するかを慎重に検討し, 近傍サイズの縮小と探索の効率化によって, 高性能化に努めた.

開発されたエンジンは, 代表的なベンチマーク問題を利用して, 既存のアルゴリズムとの性能比較を行っている. しかし, それぞれの標準問題はかなり汎用性の高い定式化に基づいているため, 総合的な評価は困難であることが多い. 通常, 計算結果が報告されているのは, 標準問題のある特殊な場合について, その専用アルゴリズムによる結果ということが多いからである. そこで, 当面の目標として, それら専用アルゴリズムと比べて遜色ない結果を得ることとし, 多くの場合, この目標をほぼ達成できたと考えている. それぞれのエンジンの汎用性の高さを考慮すると, 現実の応用においても, かなり使えるレベルに達していると言ってよかろう.

なお, 開発されたエンジンの詳細な記述と計算結果は関連文献 [20, 21, 22, 12, 14, 33, 32, 34] にある. また, これらの研究は一部文部省特定領域研究「アルゴリズム工学」および科学研究費基盤研究 (B) として進められ, 全体が報告書 [11] にまとめられている. 次節では, 問題解決エンジンの一つである, RCPSP (資源制約スケジューリング問題) についてやや詳しく述べる.

5 資源制約スケジューリング問題のエンジン

スケジューリング問題は応用において重要であるため, 1950 年代以降, 多くのモデルが定義され, その複雑さの解明や効率の良いアルゴリズムの開発がなされてきた [4]. しかし一口にスケジューリングと言っても, 生産スケジューリング, 時間割作成, 配送計画, 要員計画など様々なものがあり, 実際の応用では, 個々の状況に応じた特殊な制約なども考慮しなくてはならない. 従来の研究では, これら多様なスケジューリング問題のそれぞれに着目し, 専用のアルゴリズムを開発するという視点が重視されてきた. これに対し, 我々のアプローチは, 専用アルゴリズムではなく, 様々なスケジューリング問題を包括的に扱うことのできる汎用スケジューラの開発である.

この目的に, 資源制約スケジューリング問題 (RCPSP) を標準問題として採用する. この問題は汎用性が高く, ジョブショップ問題など, 代表的なスケジューリング問題はもとより, 現実に現れるほとんどの問題を扱うことができる. RCPSP に関してすでに多くの研究があり, アルゴリズムも開発されてきた [2, 3, 6, 9, 10, 15, 17, 18, 26, 27, 28]. 我々の定式化では, 汎用性をさらに高めるための工夫を加えている. 以下, 問題の定式化についてやや詳しく説明した後, エンジンの内容と性能について簡単に述べる. これらの詳細は [22] にある. 開発されたエンジンは, すでにいくつかの企業のスケジューリング問題に適用され, システムに組み込まれて日常的に利用されているものもある.

5.1 RCPSP による定式化

RCPSP とは, 一口で述べると「有限資源の下で複数の仕事を処理するにあたり, 作業の開始時刻と資源配分を決定する問題」であり, その基本要素は資源と作業である. 資源はさらに,

> **再生可能資源**: 機械や人, 作業場所など, 各単位時間 (e.g., 1 日) に, (前時間までの消費量とは無関係に) 予め決められた量が供給される資源,
>
> **再生不可能資源**: 原材料や予算など, 単位時間あたりではなくスケジュール全体を通して利用できる量が決まっている資源,

に分類される. どの作業が, どの資源をどれだけ消費するかは既知であるとする. 各資源に対して, その消費量の合計は供給量を越えてはならないという資源制約を課すことができる.

しかし, 例えば「通常 1 人で 2 日かかる作業が, もう 1 人臨時に雇うことで 1 日でできる. ただし, そのときコストが発生する」などの状況も現実にはよくある. 本システムでは, 作業の処理方法を表すモードを導入し, モードの選択を許すことでモデルの柔軟性を高めている.

以上を一般的に記述する. すなわち, 資源集合 $\mathcal{R} = \mathcal{R}^{\mathrm{re}} \cup \mathcal{R}^{\mathrm{non}}$, 作業集合 $\mathcal{J} = \{1, 2, \ldots, J\}$, および処理モード集合 $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_J$ が与えられる. 上に述べたように, 資源集合 $\mathcal{R}$ は再生可能資源集合 $\mathcal{R}^{\mathrm{re}}$ と再生不可能資源集合 $\mathcal{R}^{\mathrm{non}}$ に分割されている. 再生可能資源 $r \in \mathcal{R}^{\mathrm{re}}$ は各単位時間 $[t-1, t)$, $t = 1, 2, \ldots,$ に利用できる量 K^{re}_{rt} がそれまでのスケジュールに依らないのに対し, 再生不可能資源 $r \in \mathcal{R}^{\mathrm{non}}$ ではスケジュール全体を通して利用できる量 K^{non}_{r} が決まっている. また, 作業の処理方法を表す処理モードそれぞれに対して, 処理時間および処理に必要な資源量が決まっている. 各作業 j はそれに対応する処理モードの集合 $\mathcal{M}_j$ に属するいずれか一つの処理モードで処理される. 作業 j が処理モード m_j で処理されるとき, 処理時間 p_{m_j} と作業の処理に必要な資源集合 $\mathcal{R}_{m_j} \subseteq \mathcal{R}^{\mathrm{re}}_{m_j} \cup \mathcal{R}^{\mathrm{non}}_{m_j}$ が指定される. 処理に必要な資源量は, 再生可能資源 $r \in \mathcal{R}^{\mathrm{re}}$ に対しては, 処理開始後各単位時間ごとに $k^{\mathrm{re}}_{m_j ru}, u = 1, 2, \ldots, p_{m_j}$, 再生不可能資源 $r \in \mathcal{R}^{\mathrm{non}}$ に対しては $k^{\mathrm{non}}_{m_j r}$ である.

スケジュールは, 各作業 j の処理モード $m_j \in \mathcal{M}_j$, および開始時刻 s_j を用いて, $(m, s) = ((m_j \mid j \in \mathcal{J}), (s_j \mid j \in \mathcal{J}))$ で表される.

なお, 本 RCPSP モデルでは, 各再生可能資源の供給量が時間に関し一定である必要はない. これにより, 「ある期間, 機械を停止しなくてはならない」「週末は平日よりも出勤する作業員が少ない」「毎週決められた曜日にしか処理することのできない作業がある」など, より現実的な状況を記述することが可能となる. さらに, 各作業が消費する資源量も処理中一定である必要はない. したがって, 「作業を開始時には人手を必要とするが, 後は機械が勝手に処理してくれる」といった状況にも対応できる. また, 間を開けずに連続して処理しなくてはならない複数作業を, 1 つのまとまった作業として扱うことも可能である.

本 RCPSP モデルでは, 資源制約の他にも,

> **先行制約** $i \prec j$: 作業 i の処理が完了するまで作業 j の処理を開始してはならない,
>
> **再生型資源** r **上の排他的先行制約** $i \prec\prec_r j$: 作業 i は作業 j に先行し $(i \prec j)$, さらに i, j がともに再生型資源 r を消費するならば, i の処理完了後, 資源 r 上において i の次に処理される作業は j でなくてはならない,

を加えることができる. この排他的先行制約を用いると, 段取り替え作業を扱うことが可能になる.

また, 実社会に現れるスケジューリング問題の多くは独自の制約をもつため, ユーザ定義の制約をアドホックに追加できることが望ましい. その目的に, 各作業 j の開始時刻 s_j, お

よび j のモードを表す 0-1 変数

$$x_{jm} = \begin{cases} 1, & \text{作業 } j \text{ がモード } m \text{ で処理される,} \\ 0, & \text{その他,} \end{cases}$$

に関する任意の線形不等式を, ユーザ定義の制約として追加することができる.

定式化例: 勤務時間を考慮した生産スケジューリング

このスケジューリング問題 [19] は,「n 個の仕事を m 台の機械上で処理する」,「各仕事は複数の作業から成り, それらをどの順序で, どの機械上で処理するかは予め与えられている」というジョブショップ型のスケジューリング問題に加え,

(a) 各作業は, その処理に人手を要する「手動」部分 (前半) と, 機械が自動処理を行う「自動」部分 (後半) から成る,

(b) 手動部分の作業員の数は機械台数よりも少ない,

(c) 仕事の切り替えには段取り替え作業が必要である,

(d) 昼休みや休憩の間は, 手動部分の処理を中断しなくてはならない (自動部分は続行してよい),

(e) 作業の割り込みは許さない (手動部分の処理中断後, 同じ作業を引き続き処理しなくてはならない),

といった特徴がある. このうち (a)(b) は,「作業員」資源を導入することで対処できる. (c)(e) も排他的先行関係を用いて処理できる. 厄介なのは (d) である. ここでは, 各作業の「手動」部分を, ある一定時間 T (例えば 30 分) の部分作業に分割して考える (図 4 参照). これにより, (部分作業の継ぎ目に限り) 作業の中断を実現することができる. 分割された部分作業は (他の作業あるいは部分作業が割り込むことなく) 連続して処理しなくてはならないが, これは, 連続する部分作業 「手動 1」「手動 2」$\cdots$「自動」間に, 排他的先行制約「手動 1 $\prec\prec_r$ 手動 2」,「手動 2 $\prec\prec_r$ 手動 3」, $\cdots$ を加えることで対処できる. このアプローチにより, 20 仕事 29 機械 (作業数 161, 分解後の部分作業数約 3000) の問題例を解いたところ, 現状のシステムによる解と比べて, 最大完了時刻を 1 割ほど短縮することができた (図 5 参照).

Figure 4: 手動部分の分割

5.2 RCPSP エンジン

RCPSP のスケジュールは, 上述のように (m, s) で与えられるが, その探索には, 作業のリスト $\pi = (\pi(1), \pi(2), \ldots, \pi(J))$ に基づく表現 (m, π) を用いるのが便利である. 各 $\pi(j)$ は一つの作業であり, π はすべての作業の順列となっている. すなわち, π で指定された順に, 各作業 $\pi(j)$ (そのモードは $m_{\pi(j)}$) の最早開始時刻を次々と設定することで, スケジュール (m, s) を決定する. $\pi(j)$ の最早開始時刻を計算するには, 作業間の先行制約 $i \prec j$ と排他的先行制

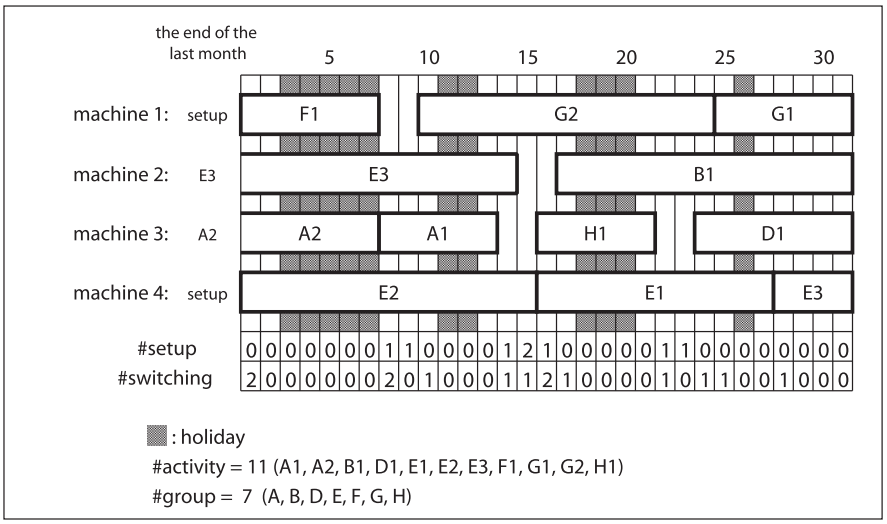

Figure 5: 手動部分の中断を許すスケジュール

約 $i \prec\prec_r j$, さらに資源制約を考慮しなければならない. とくに, 排他的先行制約を満たすために, すでにスケジュール済みの作業の開始時刻を遅らせる必要が生じることがあり, バックトラックを含んだ反復計算となる. 以上のアルゴリズムを CONSTRUCT と呼ぶ.

スケジュールの元になる (m, π) の探索は, タブー探索に基づいて行うが, 基本となる局所探索の性能を高めるため, 種々の工夫を加えている. その基本である局所探索の近傍には,

1. モードベクトル m の一つの要素を変更する.
2. 順列 π において, 作業 $\pi(j_1)$ を作業 $\pi(j_2)$ の後に移動させる.
3. 順列 π において, 作業 $\pi(j_1)$ を作業 $\pi(j_2)$ の前に移動させる.

などの操作によって実現されるものを採用している. しかし, 可能なすべての操作を許すと, 近傍サイズが大きくなり, 探索の性能が落ちるので, まず先行制約 $i \prec j$ と $i \prec\prec_r j$ に矛盾を生じないかどうかをチェックし, さらにその中で, 現在破られている考慮制約のどれかを改善するものに限定している. こうすることによって, 無駄な探索をあまり含まない近傍探索が実現されている.

5.3 性能評価

RCPSP エンジンの性能を評価するために, ベンチマーク問題に対する計算実験を行った [22]. その結果の一部を紹介する. 使用コンピュータはワークステーション SUN Ultra2 (300MHz, 1Gbyte メモリ) である.

代表的なスケジューリング問題として知られるジョブショップ問題は, RCPSP の特殊な場合として定式化できる. この問題に対しては専用アルゴリズムが多数開発されてきた. 表 1 は Fisher と Thompson [7] による有名なベンチマーク問題である ft10 と ft20 に対する結果である. ft10 は 10 仕事 × 10 機械 (100 作業) の問題例, ft20 は 20 仕事 × 5 機械 (100 作業) である. どちらに対しても 300 秒 × 30 回の計算を行い, 得られた完了時刻の最良値, 平均値, 最悪値を示している. これらについては最適値が既知であるので, その値も記入してあるが, 比較的良い解が得られていることが分かる.

つぎに, 一般の RCPSP の問題例 [16] の中から, 表 2 にある 5 つのタイプについて計算実験を行った. 結果を表 3 と表 4 に示す. 前者は, タイプ j30.sm の 480 個の問題例についての

Table 1: Computational results for ft10 and ft20.

instances	optimum values	our results		
		min	average	max
ft10	930	938	945.7	960
ft20	1165	1181	1202.5	1218

Table 2: PSPLIB の問題例

instance type	#activities	#modes	#renewable resources	#nonrenewable resources
j30.sm	30	1	4	0
j60.sm	60	1	4	0
j90.sm	90	1	4	0
j120.sm	120	1	4	0
j30.mm	30	3	2	2

結果である. これらについては最適値が既知であるので, 1 秒と 2 秒の計算時間によって得られた値の最適値からの相対誤差 (%) を示し, RCPSP に対する他の 2 種のアルゴリズムと比較している. 後者は残りのタイプに対する結果であるが, これらについては最適値が得られていないものが多いので, 知られている最良の結果と同等以上の結果が得られた問題例の個数を示している. そこにあるように, この計算によって従来の最良の解がさらに更新された問題例も多い. この点を詳しく見るため, さらに計算時間を増やし, 最良解の改善が得られるかどうかを調べた. 表 5 はその結果である.

6 今後の課題

標準問題によるアプローチで最も難しいところは, 解決すべき対象をどのようにモデル化し, どの標準問題に定式化するかという部分であろう. CSP と RCPSP に限っても, 問題の構造と制約条件をよく理解しなければ, どちらが適しているかを判断できないことがある. 場合によっては, VRP や SCP など, 他の標準問題が適当であるかもわからない.

現状では, 標準問題のアルゴリズムはすべて単体として開発されており, 全体を「問題解

Table 3: タイプ j30.sm の計算結果

	BBK[1]	H[2]	ours	
average errors [%]	0.4	0.24	0.27	0.18
average CPU time [sec]	15	4.00	1.00	2.00
maximum CPU time [sec]	108	4.00	1.00	2.00

1. Baar, Brucker and Knust (1998): on a workstation Sun Ultra 2 (167MHz)[2]
2. Hartmann (1998): on a personal computer with a Pentium processor (133MHz)[9].

Table 4: PSPLIB ベンチマークの計算結果

instance type	#instances	#instances whose optimum or best known values are found[1]	#iterations per run	ave. CPU time per run [sec]
j60.sm	480	371 (1)	10000	26.5
j90.sm	480	370 (10)	30000	181.3
j120.sm	600	219 (52)	30000	641.7
j30.mm	550	392 (66)	10000	33.6

1. (): The number of instances whose best known values were improved by our code.

Table 5: 最良解の改善をみた問題例数

instance type	#best solutions improved by our code[1]
j60.sm	6
j90.sm	17
j120.sm	107
j30.mm	134

決システム」にまとめ上げるための研究と作業が残されている. すなわち, 問題のデータを一元化して共通のフォーマットで入力できること, どの標準問題が適しているかを判断し本質を簡明に捉えた定式化を可能にすること, アルゴリズムの性能を最大限引き出せるようにその利用指針を提供すること, 得られた結果を理解しやすい形に表示し解析を容易にすること, など多くの課題がある.

これらの課題を克服できたとしても, 最後には, 現場の担当者と, アルゴリズム側が協力して問題の理解をはかり, その解決のためにどのように「問題解決エンジン」を利用するかを決定する, というフェーズが残る. 現場からのフィードバックに基づいて, 「問題解決エンジン」のさらなる改良も必要であろう.

謝辞

本研究, とくに標準問題のエンジンの開発は, 京都大学において研究室のメンバーであった柳浦睦憲 (現在, 名古屋大学), 野々部宏司 (現在, 法政大学) の両氏と梅谷俊治氏（現在, 電気通信大学）, 今堀慎治氏（現在, 東京大学）をはじめとする当時の学生たちによって推進されてきた. これらの人々の熱意と努力に深く感謝したい. なお, 本研究は一部, 文科省科学研究費の援助を受けて行われたものである.

References

[1] E. Aarts and J. K. Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley-Interscience, Chichester, 1997.

[2] T. Baar, P. Brucker and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem", in [29], 1-18, 1998.

[3] C. E. Bell and J. Han, "A new heuristic solution method in resource-constrained project scheduling", *Naval Research Logistics* Vol. 38, 315-331, 1991.

[4] P. Brucker, Scheduling Algorithms, Springer, 1995.

[5] E. K. Burke and G. Kendall (eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, 2005.

[6] J-H. Cho and Y-D. Kim, "A simulated annealing algorithm for resource constrained project scheduling problems", *Journal of the Operational Research Society* Vol. 48, 736-744, 1997.

[7] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules", in: J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[8] F. Glover and G. A. Kochenberger (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, 2003.

[9] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling", *Naval Research Logistics* Vol. 45, 733-750, 1998.

[10] W. Herroelen, B. De Reyck and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments", *Computers and Operations Research* Vol. 25, 279-302, 1998.

[11] 茨木,「メタヒューリスティクスによる汎用問題解決システムの構築」, 平成 13-15 年度科学研究費基盤研究成果報告書, 2004.

[12] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno and M. Yagiura, Effective local search algorithms for routing and scheduling problems with general time window constraints, *Transportation Science* Vol. 39, 206-232, 2005.

[13] T. Ibaraki, K. Nonobe and M. Yagiura (eds.) Metaheuristics: Progress as Real Problem Solvers (MIC'03), Springer, 2005.

[14] S. Imahori, M. Yagiura and T. Ibaraki, "Local search algorithms for the rectangle packing problem with general spatial costs", *Mathematical Programming* Vol. B97, 543-569, 2003.

[15] R. Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation", *European Journal of Operational Research* Vol. 90, 320-333, 1996.

[16] R. Kolisch and A. Sprecher, "PSPLIB – A project scheduling library", *European Journal of Operational Research* Vol. 96, 205-216, 1997.

[17] J-K. Lee and Y-D. Kim, "Search heuristics for resource constrained project scheduling", *Journal of the Operational Research Society* Vol. 47, 678-689, 1996.

[18] M. Mori and C. C. Tseng, "A genetic algorithm for multi-mode resource constrained project scheduling problem", *European Journal of Operational Research* Vol. 100, 134-141, 1997.

[19] S. Morito, J. Imaizumi and J.W. Park, "A mathematical programming approach to a tightly constrained scheduling problem", *Proceeding of the Production Scheduling Symposium '96* 85-90, 1996 (in Japanese).

[20] K. Nonobe and T. Ibaraki, "A tabu search approach to the CSP (Constraint Satisfaction Problem) as a general problem solver," *European Journal of Operational Research* Vol. 106, 599-623, 1998.

[21] K. Nonobe and T. Ibaraki, "An improved tabu search method for the weighted constraint satisfaction problem", *INFOR* Vol. 39, 131-151, 2001.

[22] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem", in [25], 557-588, 2002.

[23] I. H. Osman and J. P. Kelly (eds.), Meta-Heurisitcs: Theory and Applications (MIC'95), Kluwer Academic Publishers, 1996.

[24] J. H. Patterson, "A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem", *Management Science* Vol. 30, 854-867, 1984.

[25] C. C. Ribeiro and P. Hansen (eds.), Essays and Surveys in Metaheuristics (MIC'99), Kluwer Academic Publishers, Boston, 2002.

[26] S. E. Sampson and E. N. Weiss, "Local search techniques for generalized resource constrained project scheduling problem", *Naval Research Logistics* Vol. 40, 665-675, 1993.

[27] P. R. Thomas and S. Salhi, "A tabu search approach for the resource constrained project scheduling problem", *Journal of Heuristics* Vol. 4, 123-139, 1998.

[28] M. G. A. Verhoeven, "Tabu search for resource-constrained scheduling", *European Journal of Operational Research* Vol. 106, 266-276, 1998.

[29] S. Voss, S. Martello, I. H. Osman and C. Roucairol (eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization (MIC'97), Kluwer Academic Publishers, Boston, 1999.

[30] S. Voss and D. Woodruff, Optimization Software Class Libraries, Kluwer Academic Publishers, Boston, 2002.

[31] 柳浦, 茨木, 「組合せ最適化 –メタ戦略を中心として–」, 朝倉書店, 2001.

[32] M. Yagiura and T. Ibaraki, "Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation", *Journal of Heuristics* Vol. 7, 423-442, 2001.

[33] M. Yagiura, T. Ibaraki and F. Glover, "An ejection chain approach for the generalized assignment problem", *INFORMS Journal on Computing* Vol. 16, 133-151, 2004.

[34] M. Yagiura, M. Kishida and T. Ibaraki, "A 3-flip neighborhood local search for the set covering problem", *European Journal of Operational Research* Vol. 172, 472-499, 2006.