修 士 論 文

搬送計画問題に対する
ネットワーク理論を利用したアプローチ

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

千葉 英史

2003 年 3 月

修 士 論 文

搬送計画問題に対する
ネットワーク理論を利用したアプローチ

指導教官　浅野哲夫 教授

審査委員主査　浅野哲夫 教授
審査委員　中野浩嗣 助教授
審査委員　金子峰雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

110078 千葉 英史

提出年月: 2003 年 2 月

概 要

搬送計画問題とは, 他品種製造ラインにおける最適な搬送計画を求める問題であるが, 製造の担当者にとっては切実な問題である. 本論文では, この問題を最適化問題及び決定問題として定式化する. まず, ビンパッキング問題が搬送計画問題に多項式時間帰着可能であることを利用して, この問題がNP 完全であることを証明する. 次に, ある種の強い制約を加えた搬送計画問題が多項式時間で解けることを示す. 具体的には, この制約を逆に利用して, 入力サイズ n の一次元コンパクション問題に変換すると, ネットワーク理論を利用して $O(n\log n)$ 時間で解くことができる. アルゴリズムライブラリ LEDA によるアルゴリズムの実装も行った.

目 次

第1章　はじめに

1.1　背景

搬送計画問題は, 長い間, 製造の担当者が抱えていた比較的ローカルな問題であった. しかし, 徐々に脚光を浴び, 今では製造業の IT 化の要として積極的な投資の対象になっている. その背景には従来の製造業とこれからの製造業とが, まったく違った環境の中で生きなければならないという製造業界の進展がある. 従来からのシーケンシャルな製造プロセスは, 製品固有の製造装置を準備して, 一連のプロセスを繰り返し行う. この方式は同種の製品を大量に生産するのに適しており, 長い間利用されて来た. しかし, 最近は顧客のニーズが多様化しており, 多くのオーダーメードにも対応するため, 従来型の製造プロセスに不都合が生じてきた. また, 工場内における世代交代が進み, 従来型の製造プロセスを熟知する人間がいなくなってしまったこともあり, 今新しい製造プロセスを構築しようとする動きが見られる. そのプロセスは多品種製造ライン上での話になってくるが, 多品種に変更した途端に, 最適な搬送計画を求めるのが難しくなり, 今まであまり学術的に取り上げられることが無かった. 現在, 製造装置の制御プログラムでは, 経験的に良いとされる方法で実装されていることが多いが, そのプログラムの最適性を証明するのは容易ではない.

さて, 具体的な製造装置への要求は, 決められた時間内で一定個数の製品を生産することである. もちろん生産性を向上するには, 搬送計画もより最適なものにしなければならないのだが, プログラム内の実際の搬送命令は, イベントが発生すると IF-THEN ルールで所定のステップを実行するものが大部分で, インテリジェントには出来ていない. このような明示的な命令の集まりは莫大な状態遷移図を構成し, プログラムの全体像が把握しづらいので, バグの無い安定したソフトの確実な生産に不都合である. 明示的な制御はもはや限界であり, 改めて製造装置全体を高い所から大きく見直してみる.

搬送計画問題は, 他品種製造ラインにおける最適な搬送計画を求める問題であるが, このような最適化問題は NP 困難のクラスに属する. 多くの実務上重要な問題は, NP 困難な組合せ最適化問題であり, これらの問題に対して最悪の場合の計算量が多項式時間になる厳密解法は, 理論的には絶望視されている [1]. ここで, ある種の強い制約を付け加えた問題を考えると, 線形計画問題として定式化できる. 後は, 数理計画ソフトウェアを用いて求解可能である. ただし, パッケージを使うために, 柔軟性に欠けていて, 扱いにくい面がある. 最適化問題に対する別の角度からのアプローチとしては, 最適解を求めるのをあ

きらめ, 最適解の近似解を実用的な時間で求めるものである. 近似解を求めるアルゴリズムを近似アルゴリズムと呼ぶ. ただし, 近似解の精度が理論的に保証されない場合には発見的アルゴリズムと呼ばれる. 近似アルゴリズムでは, 得られる解（近似解）の性能を理論的に保証する. 近似アルゴリズムの研究は古く, ビンパッキング問題, 最大カット問題, 充足最大化問題など, 代表的な組合せ最適化問題について多くの研究がある [2]. しかし, それらは, 問題特有の構造を利用していて, 近似アルゴリズム設計の一般的な枠組みを提供するにはいたらなかった. 最近の近似アルゴリズム設計における技術的な進展は, 特に, 数理計画法に基づいた統一的な設計法が提案されたことである. この設計法で得られた近似アルゴリズムが, 個別に開発された従来のアルゴリズムでは達し得なかった高い近似性能を達成する [3].

1.2 目的

本研究は, 実際に工場で直面しているオブジェクト搬送スケジューリング問題に対する解決策を見出すために行う. まずは, 問題の本質を見定め, 問題を数学的に定式化する. オブジェクトの状態が変化することをイベントと呼ぶとき, 最後のイベントの発生時刻が最小となる搬送計画を求めよ, という最適化問題を考えることになる. 決定問題としても定式化する. 次に, ビンパッキング問題からの帰着を利用して, 搬送計画問題が NP 完全であることを証明する. また, たとえ各オブジェクトの処理開始時刻が順番付けされたとしても, NP 完全であることを証明する. 多くの実務上重要な問題は, NP 完全問題であり, これらの問題に対して最悪の場合の計算量が多項式時間になる厳密解法は, 理論的には絶望視されている. 最後に, 全てのイベント発生が順序付けされれば, 多項式時間で解けることを示す. 具体的には, この制約を逆に利用して, 入力サイズ n の 1 次元コンパクション問題に変換する. 1 次元コンパクション問題に対する 1 つの解法としては, 線形計画法が挙げられる. 問題の条件を線形制約式として定式化できるので, 数理計画ソフトウェアを用いて求解可能である. ただし, パッケージを使うので, 柔軟性に欠けていて扱いにくい面がある. そこで, ネットワーク理論に基づく方法を提案する. 線形計画問題をネットワーク上の問題へと変換することにより, 様々なネットワークの性質を利用するところに特色を持つ. 1 次元コンパクション問題に対して, ネットワーク理論を使って解く手法は, LSI の最適レイアウト決定問題などに応用された. この手法を, 制約付き搬送計画問題に適用させ, $O(n\log n)$ 時間で解く. また, LEDA によるアルゴリズムの実装も行う. LEDA は, ドイツのザールブリュッケンにあるマックスプランク研究所で開発された C++のクラスライブラリである. 「アルゴリズム+LEDA=プログラム」と宣伝しているように, アルゴリズムが決まって, LEDA があれば, プログラムが出来てしまうという使いやすさに大きな特色がある [4]. 実際, アルゴリズムとプログラムの間の差が小さく, かつ, 30 行程度のソースコードに仕上がる.

1.3 構成

本研究は, 上述の搬送計画問題を学術的に取り上げて, 他品種製造ラインにおける最適な搬送計画を求めるために行う. 第 2 章では搬送計画問題の定義を与え, 多くの具体的な例を示す. 第 3 章ではこの定義に基づいて, 搬送計画問題が NP 完全であることを証明する. また, 各オブジェクトのスタートの順番付けをしても NP 完全であることを示す. 第 4 章ではある種の強い制約を加えた搬送計画問題に対する解法を示す. この制約を加えることによって, 入力サイズ n の単純な一次元コンパクション問題に変換されるので, ネットワーク理論を利用して $O(n\log n)$ 時間で解くことが出来る [5]. 第 5 章では, アルゴリズムの実装に関して述べる. ここでは第 4 章で示した 2 種類のアルゴリズムを実装して, そのソースコードを全て載せる. 第 6 章では, 本研究の結論と今後の課題に関して述べる.

第2章　搬送計画問題

本章では, 搬送計画問題の定式化に至る過程をできるだけ詳しく述べる. また, 多くの例題を通して問題のパターンを観察する.

2.1　搬送計画問題の概要

図 2.1 に示すように, 対象とする製造装置は入口, 出口, いくつかの処理装置, 及びいくつかのキャリアから構成される. 入口と出口はそれぞれ一つだけだが, 処理装置とキャリアの数は任意とする. 処理装置の集合を場所集合 P とし, キャリアの集合を, キャリア集合 C とする. また, オブジェクトの集合を O とする. オブジェクトとは処理をされる品物のこととである. 各オブジェクトには処理の順番が予め決められている. 具体的には入口から投入されたオブジェクトはキャリアを使って, 決められて処理装置へと搬送される. 処理装置での処理が終了したら, 再びキャリアで決められた処理装置へと搬送される. 最後にはキャリアで出口へと搬送されて, 一つのオブジェクトの処理が終了する. 本論文で扱うオブジェクトは多品種を想定しているので, 各オブジェクトごとに処理時間が異なる. また, どの処理装置もバッファを持っていないとする. 従って, ある時間において処理装置が処理できるオブジェクトの数は高々1 つである. また, 各オブジェクトごとに処理装置における滞在時間に上限と下限を設ける. 下限は実際に処理にかかる時間である. 一方, 上限を付ける理由は実際の処理装置では, 熱処理や薬品処理などをすることがあり, それらの処理後は時間の経過とともに, オブジェクトの性質も変えてしまうことがあるからである. 従って, いつまでもそれらの処理装置に滞在することが出来ない.

キャリアについては次のようなモデルを使う. すなわち, オブジェクトを運んでいない（空の状態）時には, 現在の位置に関わらず瞬時に利用することができ, かつ, オブジェクトの持ち置きに要する時間を 0 とする. このように空の状態の時にはいつでも利用できるという仮定は, ある時間におけるキャリアの位置情報は考慮に入れないことを意味する. また, キャリアのキャパシティは特に断らない限り 1 とする. 本論文では簡単のため上のようなキャリアモデルを使うことにする.

さて, ここで簡単な搬送計画の例を示す. ここでは, 図 2.1 に示した製造装置を使う. 処理するオブジェクトは 3 つで, o_1, o_2, o_3 の順番に処理される. 処理の順番はすべて, 処理装置 A, B とする. このとき搬送計画の例を図 2.2 に示す. この図は縦軸に入口, 出口, 処理装置 A, 処理装置 B, 及びキャリアをとって, 横軸に時間をとる. オブジェクト o_1 の搬送計

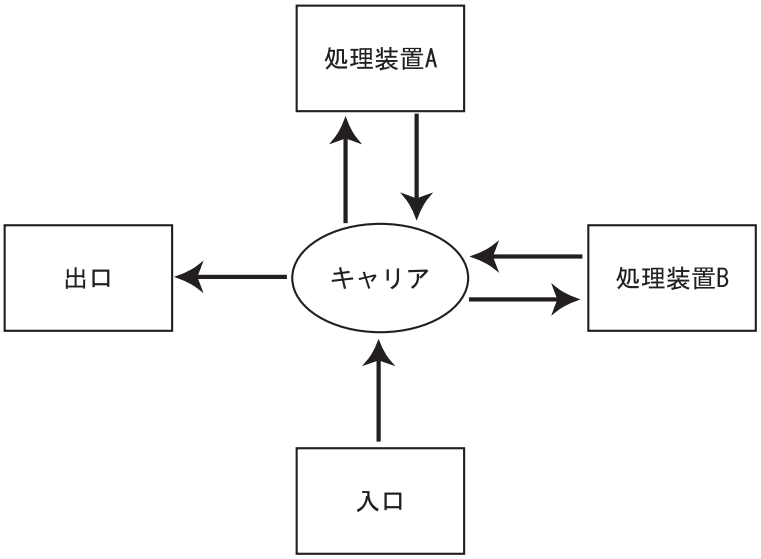


図 2.1: 簡単な製造装置の模式図

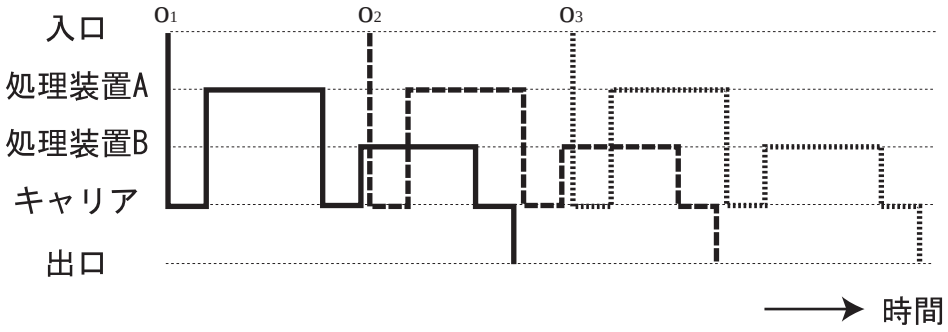


図 2.2: 搬送計画の例

画は実線で示した. 入口から提出された後, キャリアを利用して処理装置 A へ搬送される. 処理装置 A での処理が終了した後, キャリアを利用して処理装置 B へ搬送される. 処理装置 B での処理が終了した後, キャリアを利用して出口へ搬送される. オブジェクト o_2, o_3 も同様に処理される. さて, 処理装置 A, B はバッファを持っていないので, ある時点で処理できるオブジェクトの数は高々1 つであるという制約を図 2.2 は満たしている. 図 2.2 以外にも搬送計画の例は考えられる. 搬送計画問題で求めたいのは, 全体の終了時刻が最小となるような搬送計画である.

2.2 搬送計画問題の定義

本節で厳密に搬送計画問題を定義する. 搬送計画問題では, 以下の集合を扱う.

$$O = \{o_1, o_2, \cdots, o_m\} : \text{オブジェクトの集合.}$$
$$P = \{p_1, p_2, \cdots, p_l\} : \text{場所集合.}$$
$$C = \{c_1, c_2, \cdots, c_k\} : \text{キャリアの集合}$$

各オブジェクト o_i は, 一定の順序に従って場所を移動していくが, その順序を搬送順序として定義する.

$$\text{（例）} ord(o_i) = (p_1, p_3, p_4, p_1)$$

場所を移動するときには, 始点と終点の場所の対でユニークに決まるキャリアを用いる. キャリアも順序に含めたものを詳細搬送順序ということもある. たとえば,

$$ord(o_i) = (p_1, c_1, p_3, c_4, p_4, c_1, p_1)$$

のように, 場所とキャリアを交互に指定する. さて, 搬送順序は同じであってもオブジェクトごとに処理が異なることがあるので, 各場所での滞在時間も指定することにする. 本論文では, 滞在時間の上限と下限が指定されている問題を考える. キャリアについても始点と終点の位置によって時間は異なるのが一般的であるが, 本論文では簡単のため, キャリアでの滞在時間はどんな場合も 1 単位時間と仮定した. さて, 場所 p_i におけるオブジェクト o_j の滞在時間の下限と上限が l_{ji} と u_{ji} であるとき, $p_i[l_{ji}, u_{ji}]$ と表わすことにする. オブジェクト o_i の搬送順序に滞在時間の制約を加えたものを移動系列と呼び, $\sigma(o_i)$ という記号で表すことにしよう. 上の例では,

$$\sigma(o_i) = (p_1[0,0], c_1[1,1], p_3[1,4], c_4[1,1], p_4[2,5], c_1[1,1], p_1[0,0]).$$

のような系列が考えられる. 一般的には,

$$\sigma(o_i) = (p_{i_1}[l_{ii_1}, u_{ii_1}], c_{i_2}[1,1], p_{i_3}[l_{ii_3}, u_{ii_3}], c_{i_4}[1,1], \cdots)$$

のように表現することができる.

このように, 移動系列 $\sigma(o_i)$ はオブジェクト o_i が移動していく時の制約を表わしたものである. オブジェクト i の状態が変化することをイベントと呼ぶとき, イベント発生の時刻を指定すれば, そのオブジェクトの動作を完全に指定することができる. そのような系列をオブジェクト o_i の実現系列と呼ぶことにしよう. たとえば,

$$\sigma(o_i) = (p_{i_1}[l_{ii_1}, u_{ii_1}], c_{i_2}[1,1], p_{i_3}[l_{ii_3}, u_{ii_3}], c_{i_4}[1,1], \cdots)$$

の場合, オブジェクト o_i が場所 p_{i_1} に現われる時刻 t_{i_1}, o_i がキャリア c_{i_2} で移動し始める時刻 t_{i_2}, 場所 p_{i_3} に到着した時刻 t_{i_3}, $\cdots$ が全て指定することを意味する. オブジェクト o_i が場所 p_{i_k} に時刻 t_{i_k} に到着した後, 時刻 $t_{i_{k+1}}$ にキャリア $c_{i_{k+1}}$ で別の場所に移されるとき, 場所 p_{i_k} での滞在時間の下限と上限を l_{ii_k}, u_{ii_k} とすれば,

$$t_{i_k} + l_{ii_k} \leq t_{i_{k+1}} \leq t_{i_k} + u_{ii_k}$$

という不等式を満たさなければならない. このような条件を満たすオブジェクト o_i の実現系列を $\sigma_T(o_i)$ という記号で表す. $\sigma_T(o_i)$ はイベント発生時刻の系列であるから,

$$\sigma_T(o_i) = (t_{i_1}, t_{i_2}, \cdots,)$$

のような形式で表現できる.

さて, 全てのオブジェクト $o_1, o_2, \cdots, o_m$ の実現系列を全て定めたとき, それらが全体として実行可能かどうかが問題となる. ここで問題となるのは, それぞれの場所とキャリアに対して指定されたキャパシティである. すなわち, それぞれの場所とキャリアで同時に収容可能なオブジェクトの最大個数がキャパシティであり, $cap(p_i)$, $cap(c_j)$ のように表現する. 本論文では, キャパシティはすべて 1 と仮定して話を進めている.

さて, 全オブジェクトの実現系列を指定したとき, 任意の時刻 t において, 各オブジェクトが存在する場所がユニークに定まる. 場所とキャリアにはキャパシティが存在したから, どの時刻においてもキャパシティを超えるオブジェクトが同一の場所またはキャリアに存在してはならない. この制約をキャパシティ制約と呼ぶことにしよう.

結局, 最終的に求めたいのは, 滞在時間に関する制約とキャパシティ制約を満たす実現系列である.

さて, オブジェクト o_i の移動系列は一般的に,

$$\sigma(o_i) = (p_{i_1}[l_{ii_1}, u_{ii_1}], c_{i_2}[1, 1], p_{i_3}[l_{ii_3}, u_{ii_3}], c_{i_4}[1, 1], \cdots)$$

のように表現できたが, ここで,

$$p_{i_2} = c_{i_2}, l_{ii_2} = 1, u_{ii_2} = 1, p_{i_4} = c_{i_4}, l_{ii_4} = 1, u_{ii_4} = 1, \cdots$$

とおくと,

$$\sigma(o_i) = (p_{i_1}[l_{ii_1}, u_{ii_1}], p_{i_2}[l_{ii_2}, u_{ii_2}], p_{i_3}[l_{ii_3}, u_{ii_3}], p_{i_4}[l_{ii_4}, u_{ii_4}], \cdots)$$

となる. 従って, 場所集合 P とキャリア集合 C を 1 つの場所集合 P として扱うことができる. 滞在時間については, 一般的に扱うために下限値と下限値を任意とする. 以上で搬送計画問題を定式化する準備が整った. 最適化問題として見た時の本問題を以下のように定義する.

搬送計画問題 (最適化問題)

入力: 場所集合 $P = \{p_1, p_2, \cdots, p_l\}$, オブジェクト集合 $O = \{o_1, o_2, \cdots, o_m\}$, 各オブジェクト $o_i \in O$ の移動系列 $\sigma(o_i)$, 各場所 $p_i \in P$ のキャパシティ $cap(p_i)$
出力: 滞在時間に関する制約とキャパシティ制約を満たす各オブジェクトの実現系列の中で, 全体の終了時刻が最も小さいもの

次に, 決定問題として見た時の本問題を以下のように定義する.

搬送計画問題 (決定問題)

入力: 場所集合 $P = \{p_1, p_2, \cdots, p_l\}$, オブジェクト集合 $O = \{o_1, o_2, \cdots, o_m\}$, 各オブジェクト $o_i \in O$ の移動系列 $\sigma(o_i)$, 各場所 $p_i \in P$ のキャパシティ $cap(p_i)$, 正整数 T
出力: 滞在時間に関する制約とキャパシティ制約を満たす各オブジェクトの実現系列の中で, 全体の終了時刻が T 以内であるものが存在するか判定

この数学的な定義は, 搬送計画問題を厳密に記述している. 滞在時間に関する制約とキャパシティ制約を満たす各オブジェクトの実現系列が存在して, かつ, 全体の終了時刻が与えられた正整数 T 以内であれば Yes を出力する. そうでなければ No と出力する. 次の節でいくつかの具体的な問題例を与える.

2.3 搬送計画問題の例

本節では具体的な入力例を通して, 搬送計画問題を説明する. まず, 下に示した入力例を考えよう.

入力例 1：
$$
\begin{aligned}
&P = \{p_1, p_2, p_3, p_4\}, O = \{o_1, o_2, o_3\},\\
&\sigma(o_1) = (p_1[0,0], p_4[1,1], p_2[4,6], p_4[1,1], p_3[3,5], p_4[1,1], p_1[0,0]),\\
&\sigma(o_2) = (p_1[0,0], p_4[1,1], p_3[2,4], p_4[1,1], p_1[0,0]),\\
&\sigma(o_3) = (p_1[0,0], p_4[1,1], p_3[4,6], p_4[1,1], p_2[3,5], p_4[1,1], p_1[0,0]),\\
&cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = 1
\end{aligned}
$$

この入力は, 4 つの要素から成る場所集合 P と 3 つの要素から成るオブジェクト集合 O, 及び上に示した移動系列 σ で表現される. 各オブジェクトが、ある時点にどの場所にいるか（これを搬送計画という）を決めたら、集合 P の要素を縦軸に, 時間を横軸に取って図示すると分かりやすい. 入力 1 に対して, 制約条件を満たす搬送計画を図 2.3(a) に示す. オブジェクトのスタート順は o_1, o_2, o_3 となっている. 滞在時間には最小値と最大値が指定されているので, 図の方では横線が伸び縮みすることに対応している. 図 2.3(a) では, 滞在時間を全て最小にして搬送計画を立てた. このとき, 総所要時間は 18 である. 図 2.3(b) では, スタート順を o_1, o_3, o_2 とした時の搬送計画の例を示した. この例でも, 前の例と同じように, できるだけ早く終了することを考慮して, 貪欲に搬送計画を立てたものである. このとき, 総所要時間は 12 となり, 図 2.3(a) よりも短くなった. 最後にスタート順を o_3, o_1, o_2 とした時の搬送計画の例を図 2.3(c) に示す. これは総所要時間が 11 となり, 今までで最も良いものである. 実際, これは最適な搬送計画である. なぜなら, o_1 と o_3 の全体の所要時間はそれぞれ少なくとも 10 であり, また, 同時にスタートすることができないから, o_1 と

o_3 のスタート時刻を少なくとも時間 1 だけずらさなければならない. そして, 図 2.3(c) は 1 だけずらした搬送計画なので最適である.

次に, 全く同じ移動系列を複数個処理する場合を考えよう. この場合は, 同品種の搬送計画問題なので, 場合によっては貪欲法で比較的簡単に解ける.

入力例 2： $P = \{p_1, p_2, p_3, p_4, p_5\}, O = \{o_1, o_2, o_3\},$

$\sigma(o_1) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$

$\sigma(o_2) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$

$\sigma(o_3) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$

$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 1$

まず, 明らかに制約条件を満たす搬送計画を図 2.4(a) に示す. このように, あるオブジェクトの処理が完全に終わったら, 別のオブジェクトの処理を開始する, という方式を取ると制約条件を満たす搬送計画になるのは明らかである. 効率よく処理するためには, 並列的に処理すれば良い. 問題の制約条件を満たすように, 貪欲に搬送計画を立てたものを図 2.4(b) に示す. これは各場所での遊びの時間が無いので, 大変効率が良い搬送計画である. 実際の製造装置は, このような結果になる入力例が多いようである.

上の 2 つの例では, 滞在時間は全て最小値にセットしていたが, いつでも最小値を選んだ方が良いとは限らない. 下の入力例を見てみよう.

入力例 3： $P = \{p_1, p_2, p_3\}, O = \{o_1, o_2\},$

$\sigma(o_1) = (p_1[0,0], p_3[1,1], p_2[3,6], p_3[1,1], p_1[0,0]),$

$\sigma(o_2) = (p_1[0,0], p_3[5,5], p_1[0,0]),$

$cap(p_1) = cap(p_2) = cap(p_3) = 1$

明らかに制約条件を満たす搬送計画を図 2.5(a) に示した. 全体の終了時刻を早くするためには, o_1 の p_2 での滞在時間を 5 に伸ばす. そして, o_2 を中に入れると図 2.5(b) の様に最適な搬送計画になる. 最適性は全解探索から明らかである. この例のように, 滞在時間を常に最小にすれば良いとは限らないのである.

下の例も同じ主旨の入力例である.

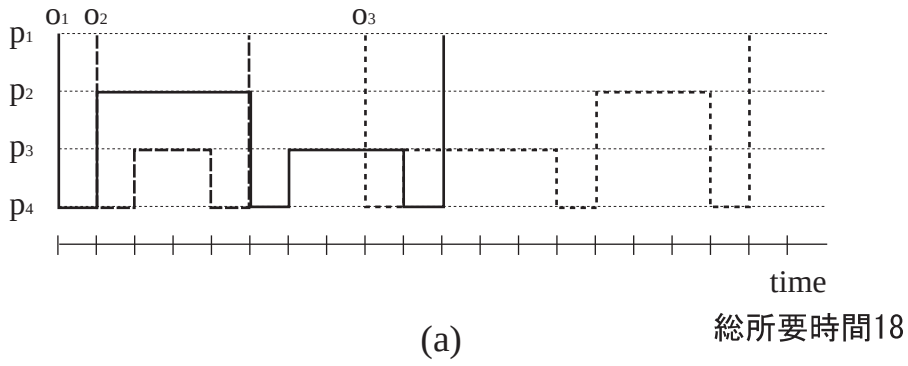
入力例 4： $P = \{p_1, p_2, p_3, p_4\}, O = \{o_1, o_2\},$

$\sigma(o_1) = (p_1[0,0], p_4[1,1], p_2[3,5], p_4[2,2], p_2[3,5], p_4[1,1], p_1[0,0]),$

$\sigma(o_2) = (p_1[0,0], p_4[2,2], p_3[1,3], p_4[1,1], p_1[0,0]),$

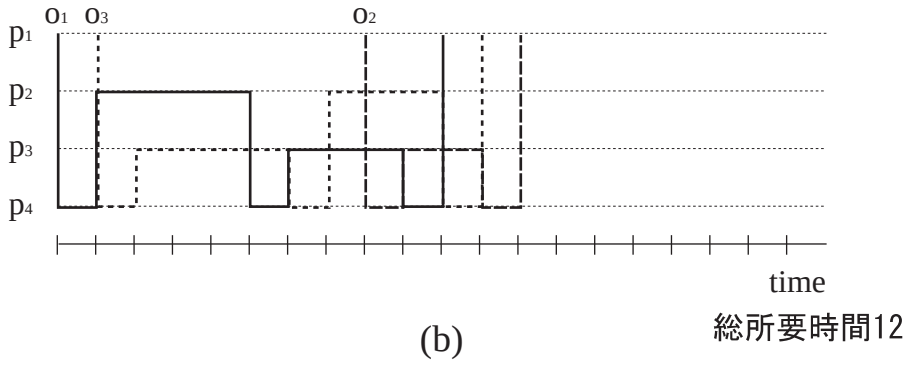
$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = 1$

明らかに制約条件を満たす搬送計画を図 2.6(a) に示す. o_2 の p_3 での滞在時間を 3 に伸ばして左に入れると, 図 2.6(b) のように最適な搬送計画となる. なぜなら, o_1 の移動系列を見ると少なくとも時間 10 はかかるが, その時間でスケジュールが終了するからである.



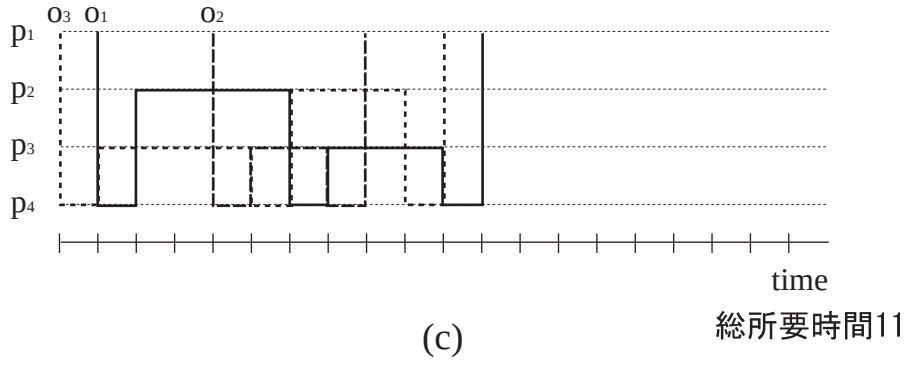
(a)

総所要時間18



(b)

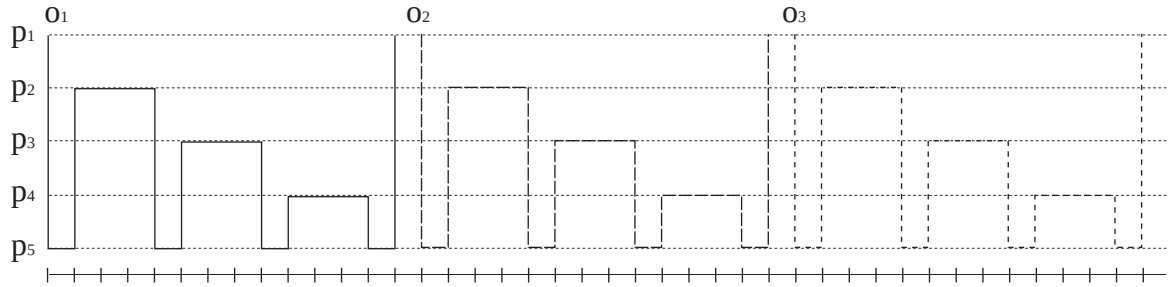
総所要時間12



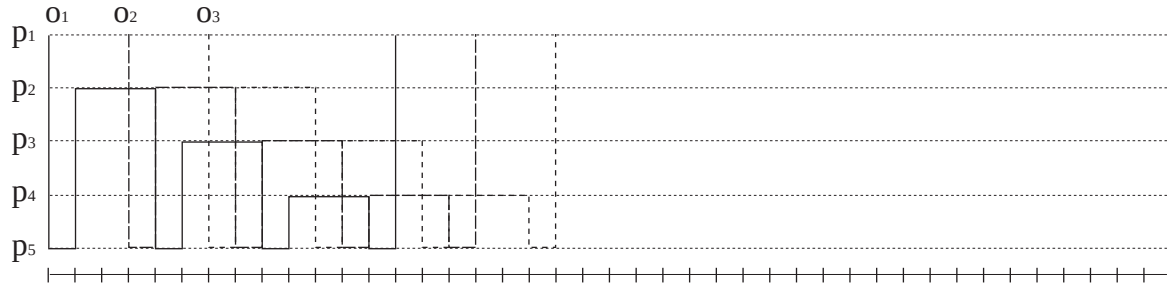
(c)

総所要時間11

図 2.3: 入力例 1 に対する搬送計画の例

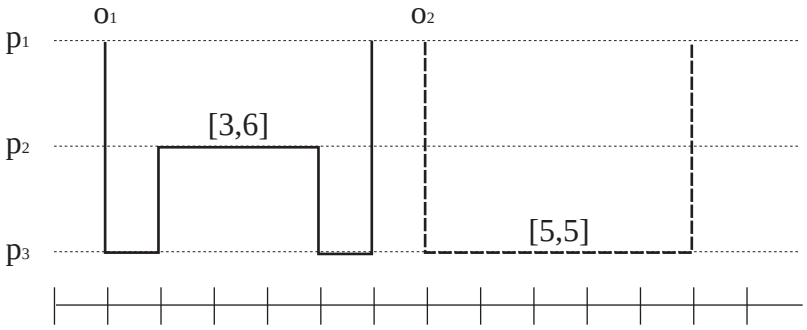


(a)

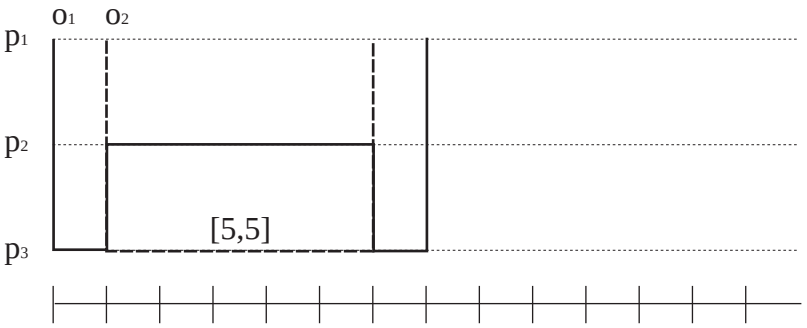


(b)

図 2.4: 入力例 2 に対する搬送計画の例

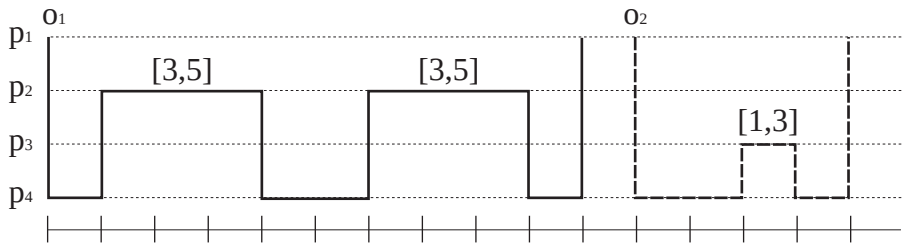


(a)

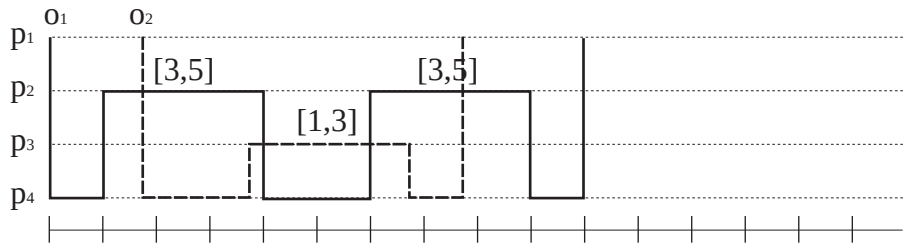


(b)

図 2.5: 入力例 3 に対する搬送計画の例



(a)



(b)

図 2.6: 入力例 4 に対する搬送計画の例

次に, 場合によっては最適な搬送計画が複数個存在することを示す.

入力例 5： $P = \{p_1, p_2, p_3, p_4, p_5\}, O = \{o_1, o_2, o_3\},$
$\sigma(o_1) = (p_1[0,0], p_5[1,1], p_2[3,6], p_5[1,1], p_1[0,0]),$
$\sigma(o_2) = (p_1[0,0], p_5[1,1], p_3[3,6], p_5[1,1], p_1[0,0]),$
$\sigma(o_3) = (p_1[0,0], p_5[1,1], p_4[2,4], p_5[1,1], p_1[0,0]),$
$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 1$

例によって, 明らかに制約条件を満たす搬送計画を図 2.7(a) に示す. また, 図 2.7(b)(c)(d) の 3 つの搬送計画はいずれも総所要時間が 7 である. したがって, 3 つとも最適である.

今まで見てきた入力例はすべての場所のキャパシティが 1 であった. 最後にそうでない例を考えよう.

入力例 6： $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}, O = \{o_1, o_2, o_3, o_4\},$
$\sigma(o_1) = (p_1[0,0], p_6[1,1], p_2[4,6], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_1[0,0]),$
$\sigma(o_2) = (p_1[0,0], p_6[1,1], p_2[4,6], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_1[0,0]),$
$\sigma(o_3) = (p_1[0,0], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_5[4,6], p_6[1,1], p_1[0,0]),$
$\sigma(o_3) = (p_1[0,0], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_5[4,6], p_6[1,1], p_1[0,0]),$
$cap(p_1) = cap(p_6) = 1, cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 2$

この入力の場合, 場所 p_2, p_3, p_4, p_5 のキャパシティが 2 なので, それぞれの場所で任意の時点で処理されるオブジェクトの数は最大 2 個まで許される. 各場所でキャパシティを満たす搬送計画を図 2.8(a)(b) に示す. (a) はオブジェクトが o_3, o_4, o_1, o_2 の順番で処理される搬送計画であり, (b) はオブジェクトが o_1, o_3, o_2, o_4 の順番で処理される搬送計画である. どちらも, 総所要時間は 18 である. 場所 p_2, p_3, p_4, p_5 のある時点でオブジェクトが 2 個処理されているのが見て分かる. 図では横線が重なっているところになる.

以上の例で見てきたように, 一般に一つの入力に対して, 複数の制約条件を満たす搬送計画を得ることができる. 本問題の難しいところは, どうやって全体の総所要時間が最も小さい制約条件を満たす搬送計画を見つけるか, ということである. 次の章で, 搬送計画問題が NP 完全であることを証明して, 問題の難しさの大枠を示す.

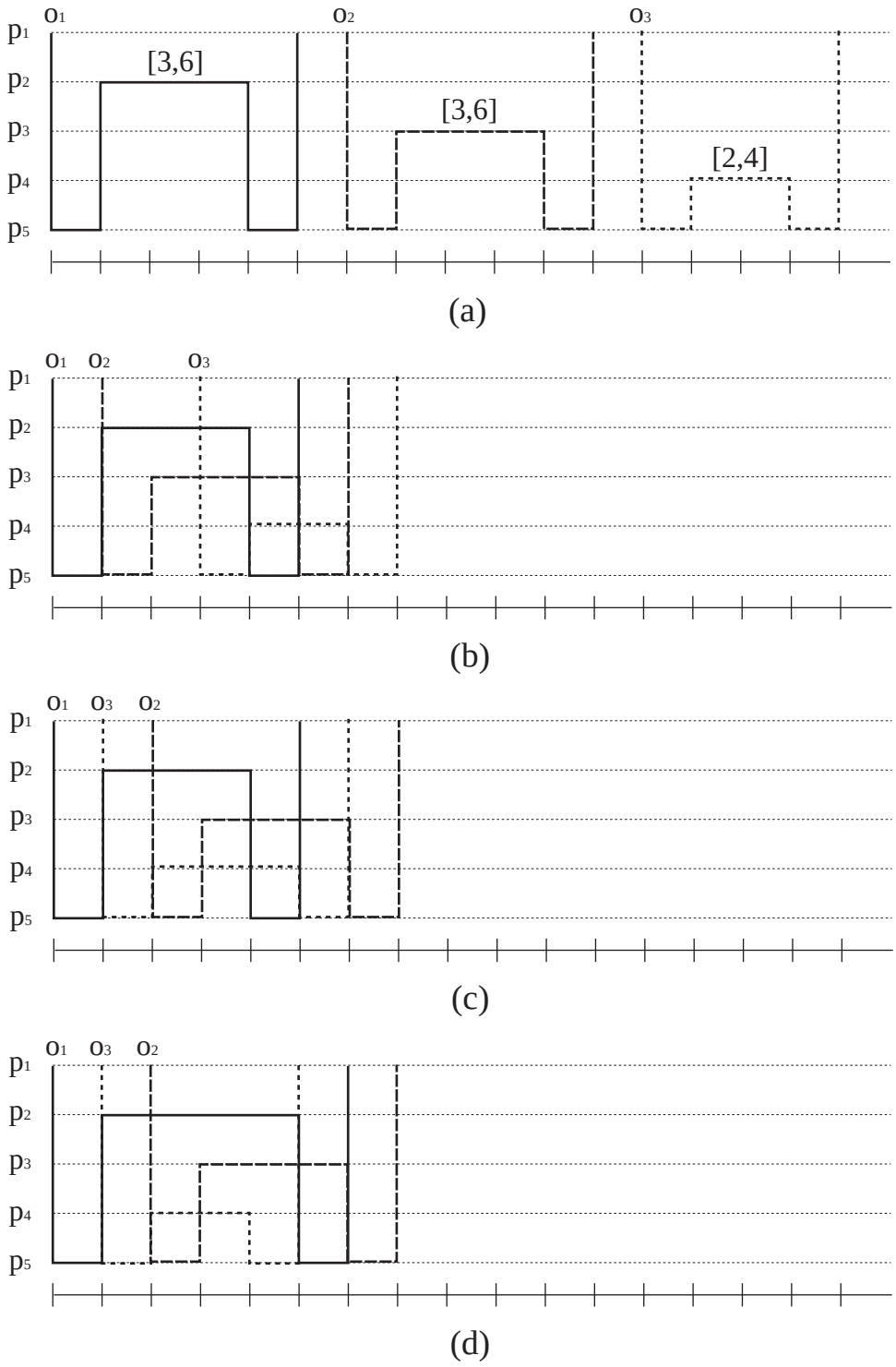
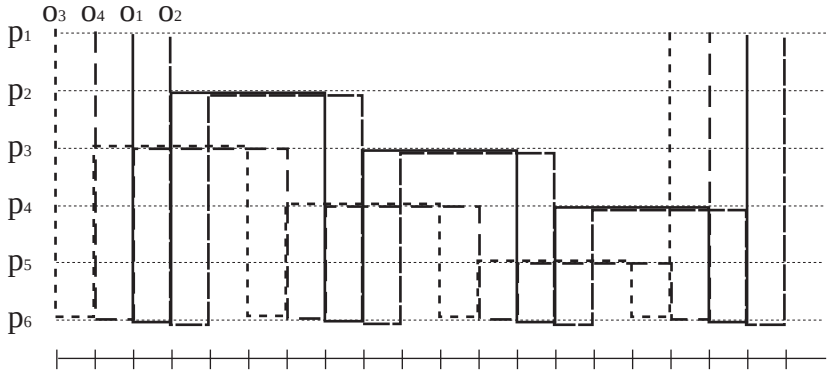
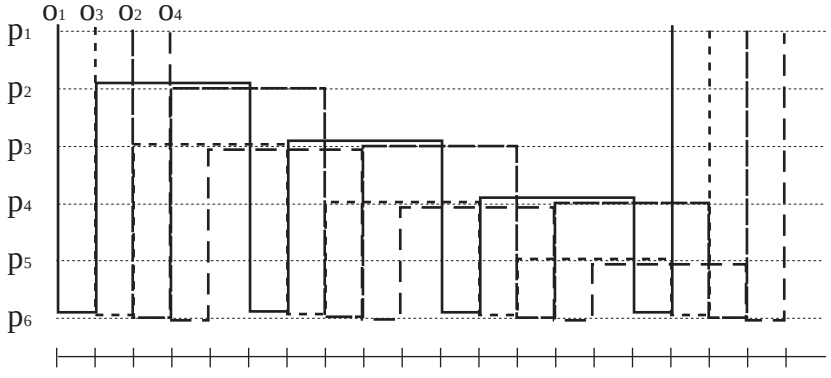


図 2.7: 入力例 5 に対する搬送計画の例



(a)



(b)

図 2.8: 入力例 6 に対する搬送計画の例

第3章　決定問題のNP完全性

P vs. NP 問題に対する一つの重要な進展は, 1970 年代初頭に Stephen Cook と Leonid Levin の仕事によってなされた. 彼らは, 個々の問題の複雑さが NP のクラス全体の複雑さと関係づけられるような, そんな問題を発見したのである. これらの問題のいずれかに対して多項式時間アルゴリズムが存在するならば, NP に属するすべての問題は多項式時間で解けるのである. こうした性質をもつ NP 問題は NP 完全とよばれる. この NP 完全性は理論的にも実用的にも重要である.

実用的な面では, NP 完全性によってある特定の問題を解くために, 存在しない多項式時間を見つけるための時間を浪費しなくてすむという利点がある. この問題が多項式時間で解けないことを証明するために必要となる数学の知識を今はもちあわせていないにもかかわらず, P は NP と等しくないと信じられているので, 問題が NP 完全であることを証明することは多項式時間で解けないことの強い根拠となっている [6].

本章で利用する定義と定理を示しておく.

定義：次の条件を満たす言語 B を NP 完全という.

1. $B \in$NP
2. NP に属するすべての A が B に多項式時間帰着可能

定理：B が NP 完全であり, かつ NP に属する C に対して $B \leq_P C$ ならば, C は NP 完全である.

本章では, 下に示した 2 つの定理を証明する.

定理 1. 搬送計画問題は NP 完全である.
定理 2. 各オブジェクトのスタートの順番に制約のある搬送計画問題は NP 完全である.

定理 1 の証明　NP 完全性の定義, 及び定理より, 以下の二つのことを示せばよい.

1. 搬送計画問題が NP に属すること
2. ビンパッキング問題を搬送計画問題に多項式時間帰着できること

まず, 搬送計画問題が NP に属することを示す. 証明のアイデアは, 搬送計画そのものを

証明書として使うことである. 検証装置へ搬送計画を証明書として与えた時, 検証装置はその搬送計画が問題の制約条件を満たしているかどうか調べる. また, 全体の終了時間が正整数 T 以内か調べる. 両方の検査に合格すれば受理する. そうでないならば拒否する. 以上で, 搬送計画問題が NP に属することを示した.

次に, ビンパッキング問題 $\leq_P$ 搬送計画問題を示す. ここで, ビンパッキング問題 (箱詰め問題 (bin packing problem)) とは代表的な NP 完全問題の一つであり, 次のように定義される.

ビンパッキング問題

入力: ビンの容量 C, ビンの個数 k, 収める物の容量の組 $(L_1, L_2, \cdots, L_n)$
出力: 次の条件を満たす $S = (S_1, S_2, \cdots, S_k)$ が存在するか判定

$$S_i \subseteq \{1, 2, \cdots, n\}, \forall i, j(i \neq j) S_i \cap S_j = \emptyset, \text{ かつ}$$

$$\bigcup_{i=1}^{k} S_i = \{1, 2, \cdots, n\}, \forall i \sum_{j \in S_i} L_j \leq C$$

n 個のアイテムと個々のアイテムに対するサイズ $L_i (i = 1, 2, \cdots, n)$ が与えられている. これら n 個のアイテムを, サイズ C のビン k 個に詰めることができるか, という問題である.

以下に示すことは搬送計画問題をサブルーチンとして使って, 上記のビンパッキング問題を解く方法である.

サブルーチン（搬送計画問題）への入力 1 :

$P = \{p_1, p_2\}, O = \{o_1, o_2, \cdots, o_n, o_{n+1}, \cdots, o_{n+k}\},$
$\sigma(o_i) = (p_1[L_i, L_i])(1 \leq i \leq n),$
$\sigma(o_i) = (p_1[1, 1], p_2[C, C], p_1[1, 1])(n + 1 \leq i \leq n + k),$
$cap(p_1) = cap(p_2) = 1,$
$T = k(C + 2)$

n 個のアイテムは, $o_i (i = 1, 2, \cdots, n)$ に対応している. そして, アイテムのサイズ $L_i (i = 1, 2, \cdots, n)$ は, オブジェクトの移動系列中の場所での滞在時間に対応している. k 個のビンは, $o_i (i = n + 1, n + 2, \cdots, n + k)$ に対応している. そして, ビンのサイズ C は, オブジェクトの移動系列中の場所での滞在時間に対応している. T の値はビンのサイズと個数から決まる定数である. この入力に対する制約条件を満たす搬送計画を図 3.1 に示した. 問題の制約条件を満たし, かつ, 全体の終了時刻が T 以内である搬送計画が存在するなら

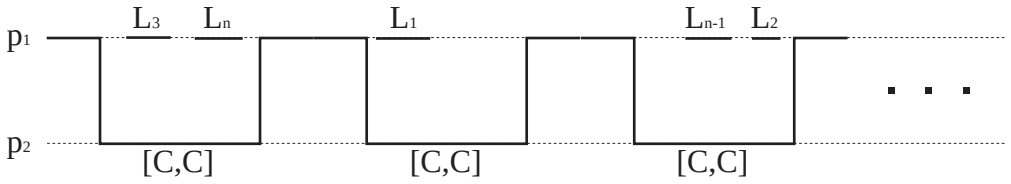


図 3.1: 入力 1 に対する搬送計画

ば, yes と答え, そうでないならば, no と答えれば良い. これで, ビンパッキング問題を解くことができる. 以上の議論から定理 1 は証明された.

定理 2 の証明　次に, 定理 2 を証明しよう. 一般の搬送計画問題にある制約をつける. 制約内容は, 各オブジェクトのスタートの順番付けである. このように制約を加えたとしても, NP 完全であることを示すことができる. 証明方法は前と同じでビンパッキング問題からの帰着による. つまり, オブジェクトのスタートが順番付けされた搬送計画問題をサブルーチンとして使って, ビンパッキング問題を解くのである. 以下にサブルーチンへの入力を示す.

サブルーチン（オブジェクトのスタートが順番付けされた搬送計画問題）への入力 2 :

$P = \{p_1, p_2, \cdots, p_{n+2}\}, O = \{o_1, o_2, \cdots, o_n, o_{n+1}, \cdots, o_{n+k}\},$
$\sigma(o_i) = (p_1[1,1], p_{i+1}[0,\infty], p_1[L_i, L_i])(1 \leq i \leq n),$
$\sigma(o_i) = (p_1[1,1], p_{n+2}[C,C], p_1[1,1])(n+1 \leq i \leq n+k),$
$cap(p_i) = 1(1 \leq i \leq n+2),$
$T = n + k(C+2)$

n 個のアイテムは, $o_i(i = 1, 2, \cdots, n)$ に対応している. そして, アイテムのサイズ $L_i(i = 1, 2, \cdots, n)$ は, オブジェクトの移動系列中の場所での滞在時間に対応している. k 個のビンは, $o_i(i = n+1, n+2, \cdots, n+k)$ に対応している. そして, ビンのサイズ C は, オブジェクトの移動系列中の場所での滞在時間に対応している. T の値はビンのサイズと個数, 及びアイテムの個数から決まる定数である. 各オブジェクトのスタートの順番は $o_1, o_2, \cdots, o_n, o_{n+1}, \cdots, o_{n+k}$ である. この入力に対する制約条件を満たす搬送計画を図 3.2 に示した. 問題の制約条件を満たし, かつ, 全体の終了時刻が T 以内である搬送計画が存在するならば, yes と答え, そうでないならば, no と答えれば良い. これで, ビンパッキング問題を解くことができる. 以上の議論から定理 2 は証明された.

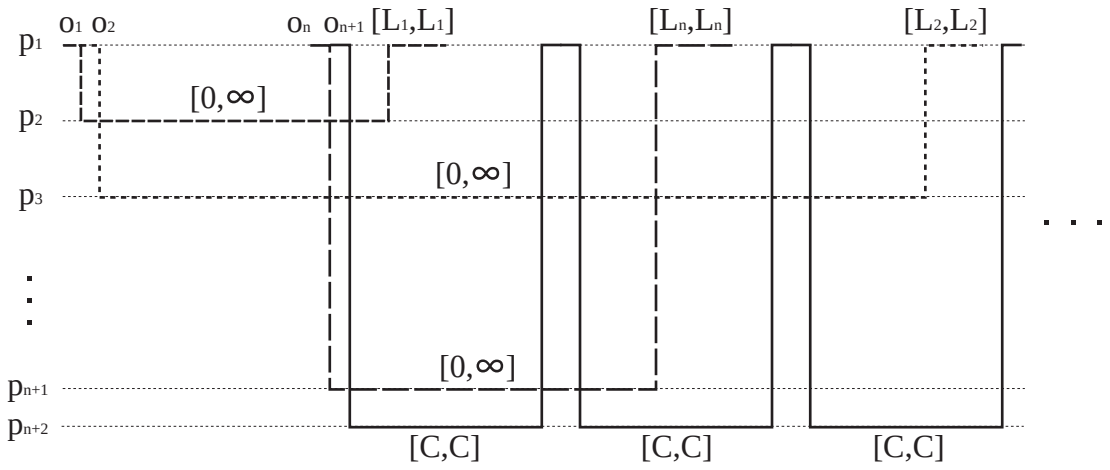


図 3.2: 入力 2 に対する搬送計画

第4章　ネットワーク理論による解法

前章で搬送計画問題は，NP 完全であることを証明した．従って, P=NP でない限り一般の搬送計画問題を完全に解く多項式時間アルゴリズムを見つけることはできない [7][8]. 本章では一般の搬送計画問題にかなり強い制約を加えて, 多項式時間で解くアルゴリズムを説明する．強い制約とは, 各場所のキャパシティを $cap(p_i) = 1(p_i \in P)$ とすることと, 全てのイベント生起の順番付けをすることである．これらの制約を加えた搬送計画問題を, 本論文では特に『制約付き搬送計画問題』と呼ぶ．この制約によって, 対象となる搬送計画の数が激減して非常に扱いやすくなる．実際, 製造装置には強い逐次性があるため, 順番付けの制約は現実問題に直結している．全てのイベント生起の順番を決めてしまえば, 単に貪欲に処理時間を減らせば良い．従って, この制約は搬送計画問題を一次元コンパクション問題に変換する．一次元コンパクション問題を解く一つの解法としては, 線形計画法が挙げられる．一次元コンパクション問題を線形計画問題として定式化することができるからである．問題の条件を線形制約式として定式化すれば, 数理計画ソフトウェアを用いて求解可能となる．ただし，パッケージを使うために，柔軟性に欠けていて扱いにくい面がある．そこで，ネットワーク理論を利用してアプローチする．これは，記述能力を高め，より高速に解くことを目指している．一次元コンパクション問題に対して，ネットワーク理論を使って解く手法は，LSI の最適レイアウト決定問題に応用された [9]．この手法を，制約付き搬送計画問題に適用させ, 多項式時間で解く．最初の節で, 線形計画法による解法に関して述べ, その次の節から, ネットワーク理論による解法に関して述べる．

4.1　線形計画法による解法

線形計画法とは, 変数に関する一次の等式や不等式で与えられた制約条件のもとで, 変数の一次関数を最小化（最大化）する解を求める方法ある [10]. 多くの問題が線形計画法として定式化でき, 実際上非常に効率的なアルゴリズムが知られている．具体的なアルゴリズムの一つはシンプレックス法である．これは, 最悪の場合には指数時間を必要とするが, 実用的には効率が良いものである．他にも線形計画問題に対する解法がいろいろとあるが, 最近は多項式オーダー性の追求が盛んに行われている [1].

線形計画問題の例を見てみよう．下の入力における搬送計画問題を考える．

入力 1：　$P = \{p_1, p_2, p_3, p_4\}, O = \{o_1, o_2\},$
$\sigma(o_1) = (p_1[0,0], p_3[1,1], p_2[3,5], p_4[1,1], p_1[0,0]),$

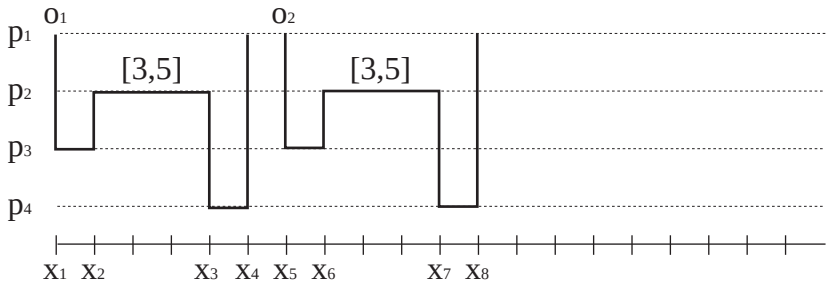


図 4.1: 入力 1 に対する搬送計画

$$\sigma(o_2) = (p_1[0,0], p_3[1,1], p_2[3,5], p_4[1,1], p_1[0,0]),$$
$$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = 1$$

この入力に対する制約条件を満たす搬送計画を図 4.1 に示す. そして, イベント生起時間に変数 $x_i(i = 1, 2, \cdots, 8)$ を割り当てる. x_8 が全体の終了時刻であり, この値をできるだけ小さくしたいので, x_8 が目的関数である.

入力より明らかなイベント生起の順番は, $x_1 \to x_2 \to x_3 \to x_4$ と $x_5 \to x_6 \to x_7 \to x_8$ である. さらに, ここでは $x_3 \to x_6, x_2 \to x_5, x_4 \to x_7$ という制約を付け加える. 距離関数 $d(i,j) = x_j - x_i$ とすると, 下の等式および不等式を満たさなければならない.

$$\begin{gathered}
d(1,2) = 1, 3 \leq d(2,3) \leq 5, d(3,4) = 1, \\
d(5,6) = 1, 3 \leq d(6,7) \leq 5, d(7,8) = 1, \\
d(3,6) \geq 0, d(2,5) \geq 0, d(4,7) \geq 0, \\
x_2 = x_1 + d(1,2), x_3 = x_2 + d(2,3), x_4 = x_3 + d(3,4), \\
x_6 = x_5 + d(5,6), x_7 = x_6 + d(6,7), x_8 = x_7 + d(7,8), \\
x_6 = x_3 + d(3,6), x_5 = x_2 + d(2,5), x_7 = x_4 + d(4,7)
\end{gathered}$$

目的関数と制約条件は全て変数の一次関数であるから, これは線形計画問題である. この線形計画問題をここでは, フリーの数理計画ソフトウェアである lp_solve を用いて解いてみた. このパッケージを使う場合, 入力形式の制約に注意しなければならない. 例えば, 『, ( )』の文字を使用できないので, 違う文字に変換する必要がある. また, 『$\leq$』と『$\geq$』は lp_solve の入力形式においてはそれぞれ『<』と『>』になる. 変数の値はデフォルトで非負なので, 0 以上という制約式を除いてもよい. lp_solve を用いて解く場合, 入力ファイルの中身は下のようになる.

$$min : x8;$$

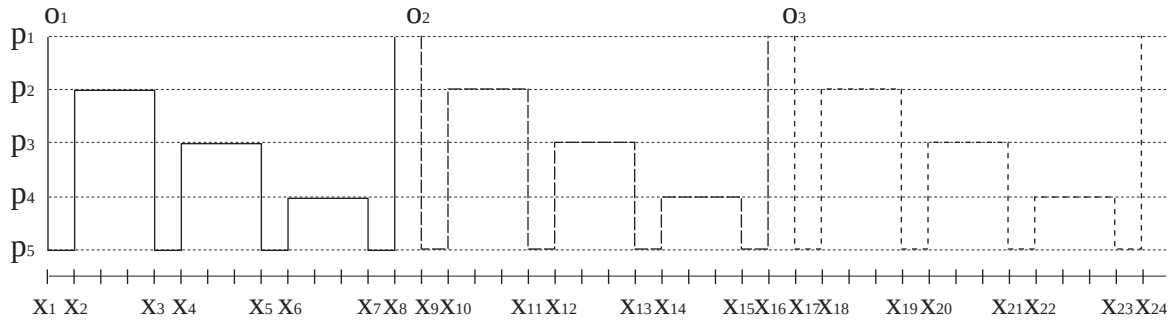


図 4.2: 入力 2 に対する搬送計画

$$
\begin{aligned}
&d1\_2 = 1; 3 < d2\_3; d2\_3 < 5; d3\_4 = 1;\\
&d5\_6 = 1; 3 < d6\_7; d6\_7 < 5; d7\_8 = 1;\\
&x2 = x1 + d1\_2; x3 = x2 + d2\_3; x4 = x3 + d3\_4;\\
&x6 = x5 + d5\_6; x7 = x6 + d6\_7; x8 = x7 + d7\_8;\\
&x6 = x3 + d3\_6; x5 = x2 + d2\_5; x7 = x4 + d4\_7;
\end{aligned}
$$

同品種の搬送計画問題の場合は, 各オブジェクトの移動系列は全く同じになる. そのような例を線形計画問題として考えてみよう. 入力を下に示す.

入力 2：
$$
\begin{aligned}
&P = \{p_1, p_2, p_3, p_4, p_5\}, O = \{o_1, o_2, o_3\},\\
&\sigma(o_1) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),\\
&\sigma(o_2) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),\\
&\sigma(o_3) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),\\
&cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 1
\end{aligned}
$$

この入力に対する矛盾の無い搬送計画を図 4.2 に示す. そして, イベント生起時間に変数 $x_i (i = 1, 2, \cdots, 24)$ を割り当てる. x_{24} が全体の終了時刻であり, この値をできるだけ小さくしたいので, x_{24} が目的関数である. イベント生起の順番付けを例えば, $x_3 \to x_{10}, x_{11} \to x_{18}, x_5 \to x_{12}, x_{13} \to x_{20}, x_7 \to x_{14}, x_{15} \to x_{22}$ とすると, 下の様に線形計画問題として定式化できる.

$$
\begin{aligned}
\textbf{min}: \quad & x_{24}\\
\\
\textbf{s.t.} \quad & d(1,2) = 1, 3 \le d(2,3) \le 5, d(3,4) = 1, 3 \le d(4,5) \le 5,\\
& d(5,6) = 1, 3 \le d(6,7) \le 5, d(7,8) = 1,\\
& d(9,10) = 1, 3 \le d(10,11) \le 5, d(11,12) = 1, 3 \le d(12,13) \le 5,\\
& d(13,14) = 1, 3 \le d(14,15) \le 5, d(15,16) = 1,
\end{aligned}
$$

$$
\begin{aligned}
&d(17,18)=1, 3 \leq d(18,19) \leq 5, d(19,20)=1, 3 \leq d(20,21) \leq 5,\\
&d(21,22)=1, 3 \leq d(22,23) \leq 5, d(23,24)=1,\\
&d(3,10) \geq 0, d(11,18) \geq 0,\\
&d(5,12) \geq 0, d(13,20) \geq 0,\\
&d(7,14) \geq 0, d(15,22) \geq 0,\\
&x_2 = x_1 + d(1,2), x_3 = x_2 + d(2,3), x_4 = x_3 + d(3,4),\\
&x_5 = x_4 + d(4,5),\\
&x_6 = x_5 + d(5,6), x_7 = x_6 + d(6,7), x_8 = x_7 + d(7,8),\\
&x_{10} = x_9 + d(9,10), x_{11} = x_{10} + d(10,11), x_{12} = x_{11} + d(11,12),\\
&x_{13} = x_{12} + d(12,13),\\
&x_{14} = x_{13} + d(13,14), x_{15} = x_{14} + d(14,15), x_{16} = x_{15} + d(15,16),\\
&x_{18} = x_{17} + d(17,18), x_{19} = x_{18} + d(18,19), x_{20} = x_{19} + d(19,20),\\
&x_{21} = x_{20} + d(20,21),\\
&x_{22} = x_{21} + d(21,22), x_{23} = x_{22} + d(22,23), x_{24} = x_{23} + d(23,24),\\
&x_{10} = x_3 + d(3,10), x_{18} = x_{11} + d(11,18), x_{12} = x_5 + d(5,12),\\
&x_{20} = x_{13} + d(13,20), x_{14} = x_7 + d(7,14), x_{22} = x_{15} + d(15,22)
\end{aligned}
$$

後は, lp_solve を使って解いてやれば良い. 下に lp_solve 用に変換した入力を示す.

$$
\begin{aligned}
&min : x24;\\
&\\
&d1\_2 = 1; 3 < d2\_3; d2\_3 < 5;\\
&d3\_4 = 1; 3 < d4\_5; d4\_5 < 5;\\
&d5\_6 = 1; 3 < d6\_7; d6\_7 < 5;\\
&d7\_8 = 1;\\
&d9\_10 = 1; 3 < d10\_11; d10\_11 < 5;\\
&d11\_12 = 1; 3 < d12\_13; d12\_13 < 5;\\
&d13\_14 = 1; 3 < d14\_15; d14\_15 < 5;\\
&d15\_16 = 1;\\
&d17\_18 = 1; 3 < d18\_19; d18\_19 < 5;\\
&d19\_20 = 1; 3 < d20\_21; d20\_21 < 5;\\
&d21\_22 = 1; 3 < d22\_23; d22\_23 < 5;\\
&d23\_24 = 1;\\
&x2 = x1 + d1\_2; x3 = x2 + d2\_3; x4 = x3 + d3\_4;
\end{aligned}
$$

$$x5 = x4 + d4\_5; x6 = x5 + d5\_6; x7 = x6 + d6\_7; x8 = x7 + d7\_8;$$
$$x10 = x9 + d9\_10; x11 = x10 + d10\_11; x12 = x11 + d11\_12;$$
$$x13 = x12 + d12\_13; x14 = x13 + d13\_14; x15 = x14 + d14\_15;$$
$$x16 = x15 + d15\_16;$$
$$x18 = x17 + d17\_18; x19 = x18 + d18\_19; x20 = x19 + d19\_20;$$
$$x21 = x20 + d20\_21; x22 = x21 + d21\_22; x23 = x22 + d22\_23;$$
$$x24 = x23 + d23\_24;$$
$$x10 = x3 + d3\_10; x18 = x11 + d11\_18;$$
$$x12 = x5 + d5\_12; x20 = x13 + d13\_20;$$
$$x14 = x7 + d7\_14; x22 = x15 + d15\_22;$$

以上で線形計画法による解法を終わる. とりあえず, この方法を使って順序の制約付き搬送計画問題を解く事ができることを示した. しかし, この解法だと lp_solve を使う場合, 入力が非常に面倒である. そこで, 次の節ではネットワーク理論を用いたアプローチに関して述べる. このアプローチは, 二つのステップから成り立っている. ステップ 1 がネットワークの構成であり, ステップ 2 がネットワークの最長路問題を解くことである.

4.2 ネットワークの構成

いくつかの点とそれらの 2 点間を結ぶ枝とによって表されるような図形をグラフという. 有効グラフ $G = (V, A)$ は点集合 V と枝集合 A の対で表現される. また, 有効グラフ $G = (V, A)$ の各枝 $a \in A$ に実数の長さ $d(a)$ が付与されているとき, ネットワーク (G, d) を G と d の対で表す.

ネットワーク理論による解法のステップ 1 はネットワークの構成である. 制約付き搬送計画問題の入力からネットワークに変換する. 入力として図 4.1 のように制約条件を満たす搬送計画が与えられた時, 図の縦線（イベント生起）をグラフの点に変換する. 図の横線（滞在時間）をグラフの枝に変換する. また, 順序の制約にも枝を対応させる.

ここで, イベント生起時間に変数 $x_i (i \in V)$ を対応させて, 枝の重み $d(i, j) = x_j - x_i$ とする時, 下の線形計画問題をネットワークを利用して解く.

$$\begin{aligned} \textbf{minimize} \quad & x_t - x_s \\ \textbf{s.t.} \quad & x_j - x_i \geq d(i,j) \;\; for \;\; each \;\; arc(i,j) \in A \end{aligned} \qquad (*)$$

ただし, 点 s と点 t はスタートとゴールを表すダミーノードとする. グラフの枝の重みは上の制約条件を満たすように付ける. 図 4.3 に図 4.1 をネットワークに変換したものを示した. 順序の制約に対応している枝は, $a(2,5), a(3,6), a(4,7)$ の 3 本である. 図 4.4 に図

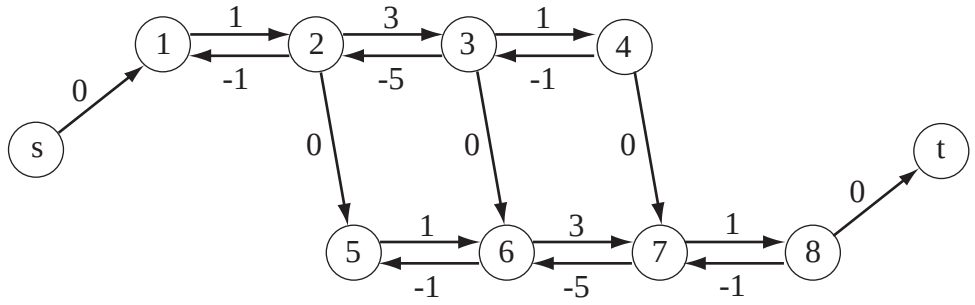


図 4.3: 図 4.1 から構成したネットワーク

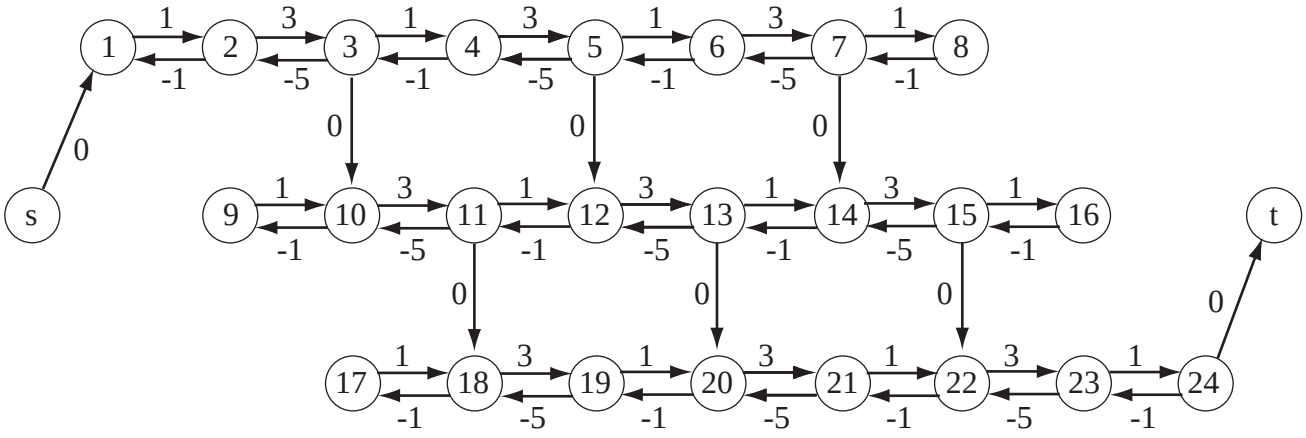


図 4.4: 図 4.2 から構成したネットワーク

4.2 をネットワークに変換したものを示した. 図 4.4 で, 順序の制約に対応している枝は, $a(3,10), a(11,18), a(5,12), a(13,20), a(7,14), a(15,22)$ の 6 本である. このように, 制約付き搬送計画問題に対する入力から一意にネットワークへと変換できる. 最後の例として以下の入力を考えよう.

入力 3： $P = \{p_1, p_2, p_3, p_4\}, O = \{o_1, o_2\},$
$\sigma(o_1) = (p_1[0,0], p_4[1,1], p_2[3,7], p_4[1,1], p_1[0,0]),$
$\sigma(o_2) = (p_1[0,0], p_4[1,1], p_3[3,5], p_4[1,1], p_1[0,0]),$
$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = 1$

この入力に対する矛盾の無い搬送計画を図 4.5 に示した. イベント生起の順序の制約として, $x_2 \to x_3, x_6 \to x_7$ を付け加える. これをネットワークに変換したものを図 4.6 に示す.

ここで, $n = |V|$, $m = |A|$ とすると, 一つの点の出次数は高々3 なので, $m \leq 3n = O(n)$ である.

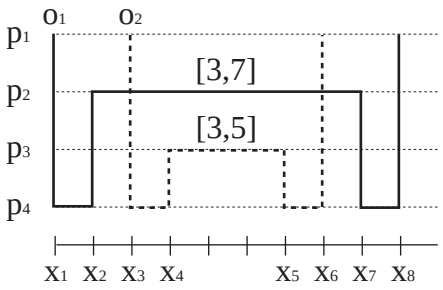


図 4.5: 入力 3 に対する搬送計画

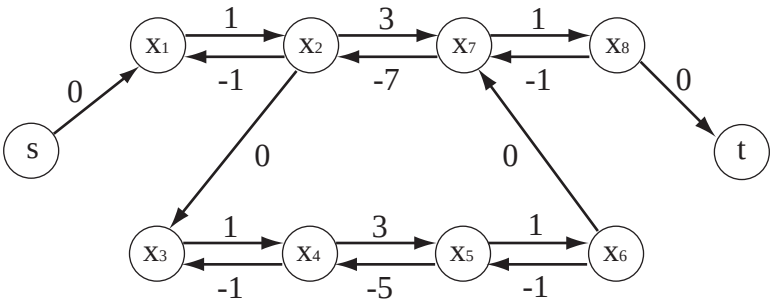


図 4.6: 図 4.5 から構成したネットワーク

4.3 ネットワークの最長路問題

前節において, ネットワークを構成したら, 次のステップで最長路問題を解く. 本来, $(*)$ の線形計画問題を解きたいのであるが, ネットワークでは点 s から点 t への最長路問題に対応している. 一般の最長路問題を下に示す.

2 頂点間の最長路問題

入力: 有向グラフ $G=(V,E)$ と各枝の長さ $d: A \rightarrow \mathbf{Z}^+$, 始点 $s \in V$, 終点 $t \in V$, 正整数 B
出力: グラフ G 中の点 a から点 b までの単純な有向道で, その長さが B 以上の有向道が存在するか判定

残念ながら, この問題は代表的な NP 完全問題の一つであり, 効率的に解くアルゴリズムは知られていない [7]. しかし, 制約条件を満たす搬送計画をネットワークに変換した場合, そこには正のサイクルが存在しない. この性質のおかげで, たとえ最長路問題でも効率的に解くことができる. (もし, 制約条件を満たさない搬送計画をネットワークに変換すると, そこには正のサイクルが存在する.)

まず, 構成したネットワーク (G,d) の枝の重みを $\forall arc(i,j) \in A, \bar{d}(i,j) = -d(i,j)$ に変える. このネットワーク $(G,\bar{d})$ においては, 最短路問題を解くことになる. 最短路問題とは下のような問題であるが, 多くの研究者により, 理論面, 実用面ともに議論され様々なアルゴリズムが提案されてきた [1]. 図 4.6 の (G,d) から構成した $(G,\bar{d})$ を図 4.7 に示す.

1 始点最短路問題

入力： 有効グラフ $G=(V,A)$ と各枝の長さ $d: A \rightarrow \mathbf{R}$, 始点 $s \in V$
出力： s から到達可能な各頂点への長さ最小の有向道とその長さ (距離) をそれぞれ出力. ただし, 始点 s から到達可能な負閉路が存在する場合は, その負閉路を 1 つ出力.

$(G,\bar{d})$ には負のサイクルが存在しない. なぜなら, (G,d) に正のサイクルが存在しなかったからである. 従って, Bellman-Ford 法を使って, 一般に $O(mn) = O(n^2)$ で解くことができる [9]. しかし, 制約条件を満たす搬送計画が入力として与えられることを利用して, $O(n\log n)$ で解くことができる.

$x_i^{(0)}$ を点 i に対応する制約条件を満たす搬送計画のイベント生起時間とする. 従って, あらかじめこの値は分かっていて, $x_i^{(0)}(i \in V)$ は, $(*)$ の制約条件を満たしている. ここで, 枝の重み

$$\tilde{d}(i,j) = x_j^{(0)} - x_i^{(0)} + \bar{d}(i,j) = x_j^{(0)} - x_i^{(0)} - d(i,j) \geq 0$$

を持つネットワーク $(G,\tilde{d})$ に変換する. 図 4.6 の (G,d) から構成した $(G,\tilde{d})$ を図 4.8 に示した. 全ての枝の重みが非負なので, ダイクストラ法が適用できる. これは, 枝の長さが全て非負の場合の 1 始点最短経路問題に対するアルゴリズムである.

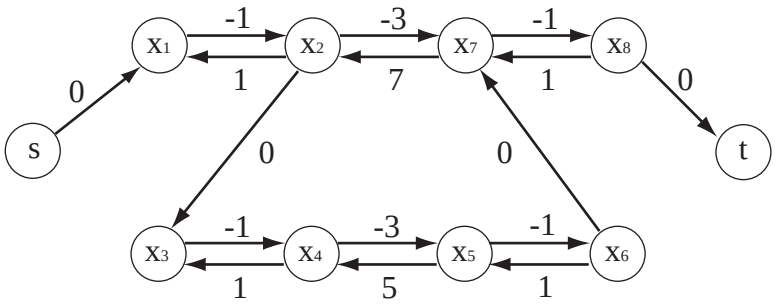


図 4.7: 図 4.6 の (G, d) から構成したネットワーク $(G, \bar{d})$

ダイクストラ法

(0) すべての頂点 v について $dist[v] = \infty$ とする.
(1) $dist[s] = 0$ とする (s は始点）.
(2) すべての頂点をプール P に蓄える.
(3) while(P が空でない){
(4) P から $dist[]$ の値が最小の頂点 u を取り出す.
(5) u にマークをつける.
(6) for(マークがついていない u の隣接頂点 v)
(7) if($dist[u] + cost(u, v) < dist[v]$)
(8) then $dist[v] = dist[u] + cost(u, v)$
(9) }

$dist[u]$: 始点 s から頂点 u までの最短経路の長さを蓄える配列.
$cost(u, v)$: 2 頂点 u, v 間の辺の重み（長さ）.
プール P: 頂点の集合を蓄えるためのデータ構造.

ダイクストラ法の時間計算量を算定する. まず, 枝の走査は, 各枝ちょうど 1 回ずつ行われるので, 全体で $O(m)$ である. 頂点の選択では, プール P の中で最小ラベルの頂点を探索しなければならない. 単純に実行すれば, 毎回, $|P|$ の手間がかかるので, 全体で $O(n^2)$ となり, ダイクストラ法は, $O(m + n^2) = O(n^2)$ 時間のアルゴリズムとなる. 頂点の選択が計算時間のネックとなるが, さまざまなデータ構造を用いて最小ラベルの頂点の選択を効率良く行い, 計算量が改良される [1]. 例えば, ヒープを使ってインプリメントすれば, $O(m\log n)$=$O(n\log n)$ で解くことができる [9]. Fredman-Tarjan の Fibonacci ヒープ [11] を用いると, $O(m + n\log n)$ となる. これは, 現在, 最良の強多項式時間である. ただし, Goldberg-Tarjan[12] はダイクストラ法における Fibonacci ヒープの平均的挙動を解析し, 実際の計算時間のほとんどは, Fibonacci ヒープよりも単純なヒープを用いた方が高速であることを裏付けている.

以上より, 入力サイズ n の制約付き搬送計画問題を $O(n\log n)$ で解けることがわかった.

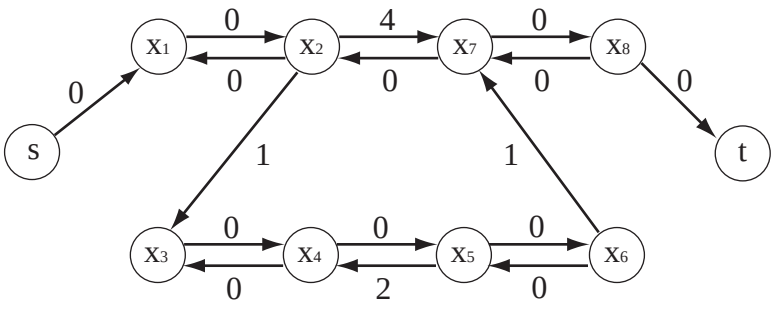


図 4.8: 図 4.6 の (G, d) から構成したネットワーク $(G, \tilde{d})$

第5章　アルゴリズムの実装

前章では制約付き搬送計画問題に対する二通りの解法を述べた．一方はBellman-Ford法，他方はダイクストラ法をそれぞれ利用したアルゴリズムであった．本章では，これらのアルゴリズムのLEDAによる実装に関して述べる．

LEDA(Library of Efficient Data types and Algorithms, リーダ) は, ドイツのザールブリュッケンにあるマックスプランク研究所で開発されたC++のクラスライブラリである. 「アルゴリズム＋LEDA＝プログラム」と宣伝しているように, アルゴリズムが決まって, LEDAがあれば, プログラムが出来てしまうという使いやすさに大きな特徴がある. また, 簡単に使えるばかりではなく, 多彩な機能と堅牢性を兼ね備えており, さまざまな用途で実務に利用されている. 実際, LEDAを利用して実装したので, アルゴリズムとプログラムとの間のギャップが小さく, かつ, 大変短いプログラムに仕上がった. 入力ファイルは図4.6のような形をしていて, あらかじめグラフウィンドウに入力となるネットワークをgw形式で作成しておく.

コンパクションプログラム
入力： $G = (V, A)$, 枝の重み d, 始点 s
出力： s から到達可能な各頂点への最長距離

以下に, Bellman-Ford法を利用したアルゴリズムの全ソースコードを載せる.

```
#include<LEDA/graphwin.h>
#include<LEDA/graph_alg.h>
int main() {
    string a;
    a = read_string("入力ファイル名 : ");
    GRAPH<int, int> G;
    G.read(a);

    node s = G.first_node();
    edge_array<int> d(G);
    d = G.edge_data();
    edge_array<int> cost(G);
    edge e;
    forall_edges(e, G)
        cost[e] = -d[e];
    node_array<int> dist(G, MAXINT);
    node_array<edge> pred(G, nil);

    bool b;
    b = BELLMAN_FORD_B(G, s, cost, dist, pred);
    if (b == false)
        cout << "負の閉路が存在する" << endl;
    else {
        node v;
        forall_nodes(v, G)
            cout << "node" << index(v) << " : " << -dist[v] << endl;
    }
    return 0;
}
```

以下に, ダイクストラ法を利用したアルゴリズムの全ソースコードを載せる.

```
#include <LEDA/graphwin.h>
#include <LEDA/graph_alg.h>
int main() {
    string a;
    a = read_string("入力ファイル名 : ");
    GRAPH<int, int> G;
    G.read(a);

    node s = G.first_node();
    node_array<int> x(G);
    x = G.node_data();
    edge_array<int> d(G);
    d = G.edge_data();
    edge_array<int> cost(G);
    edge e;
    forall_edges(e, G)
        cost[e] = x[G.target(e)] - x[G.source(e)] - d[e];
    node_array<int> dist(G, MAXINT);
    node_array<edge> pred(G, nil);

    DIJKSTRA_T(G, s, cost, dist, pred);

    node v;
    forall_nodes(v, G)
        cout << "node" << index(v) << " : " << x[v] - dist[v] << endl;
    return 0;
}
```

第6章　おわりに

6.1　まとめ

本研究は製造業で現実に直面している問題を『搬送計画問題』として定式化した. これに, ある種の強い制約を加えた問題に対して, ネットワーク理論による解法を提案した. 本研究では以下のことが明らかになった.

- 搬送計画問題はNP 完全である. これは, ビンパッキング問題が搬送計画問題に多項式時間帰着可能であることを利用して証明した. また, オブジェクトのスタートの順番付けをしてもNP 完全である. これも, 前と同じ性質を利用して証明した.
- 搬送計画問題の各場所のキャパシティを 1 にして, 全てのイベント生起の順番を決めてしまうと, 入力サイズ n の一次元コンパクション問題に変換される. これをネットワーク理論を利用して, $O(n\log n)$ 時間で解くことができる. アルゴリズムライブラリ LEDA によるアルゴリズムの実装も行った.

6.2　今後の課題

今後の課題としては，ある制約条件を満たす搬送計画に対して，新たな搬送計画を最適に挿入する方法を考案することである．この時、貪欲法や分枝限定法などを利用してアプローチする。また、本論文のモデルと別のモデルで問題を考えて見る。特にキャリアは空の時にはいつでも利用可能という仮定をしたが、実際はキャリアの現在位置によってすぐにアクセス可能とは限らない。従って、キャリアの位置情報を考慮に入れて研究する。

謝辞

本研究を進めるにあたり, 終始御指導を頂きました浅野哲夫教授に深く感謝致します. ゼミなどで大変熱心な御指導をして下さった中野浩嗣助教授, 小保方幸次助手, 元木光雄助手, ならびに浅野・中野研究室の皆様に感謝致します.

また, 本研究に関して実用的な見地から様々な御意見を頂きました大日本スクリーン製造 (株) 三塚郁夫氏を始め, 各氏に感謝致します.

参考文献

[1] 久保幹雄, 田村明久, 松井知己編集, 応用数理計画ハンドブック, 朝倉書店, 2002.

[2] D.S.Johnson, Approximation algorithms for combinatorial problems, Journal on Computer and System Sciences, Vol.9, pp.256-278, 1974.

[3] V.V. ヴァジラーニ, 浅野孝夫訳, 近似アルゴリズム, Springer, 2002.

[4] 浅野哲夫, 小保方幸次, LEDA で始める C/C++プログラミング, サイエンス社, 2002.

[5] H.Imai, Notes on the One-Dimensional Compaction Problem of LSI Layouts Viewed from Network Flow Theory and Algorithms, Trans. IECE, Vol.E69, No10, pp.1080-1083, October 1986.

[6] M. シプサ著, 渡辺治他訳, 計算理論の基礎, 共立出版, 2000.

[7] M.R.Garey and D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[8] T. コルメン, C. ライザーソン, R. リベスト著, 浅野哲夫他訳, アルゴリズムイントロダクション, 近代科学社, 1995.

[9] R.E.Tarjan, Data Structures and Network Algorithms, CBMS Regional Conference Series in Applied Mathematics, Philadelphia, 1983.

[10] 福島雅夫, 数理計画法入門, 朝倉書店, 1996.

[11] M.L.Fredman,R.E.Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of the ACM, 34:596-615, 1987.

[12] A.V.Goldberg,R.E.Tarjan, Expected performance of Dijkstra's shortest path algorithm, Technical Report 96-062, NEC Research Institute, 1996.