博 士 論 文

自動搬送システムにおける衝突確率の解析

指導教官　浅野 哲夫 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

千葉 英史

2006 年 3 月 22 日

要 旨

本研究では，液晶や半導体などの製造装置によく見られる直列型の搬送系をモデル化して，搬送系を流れるジョブ同士の衝突確率を解析する．我々のモデルでは，ジョブの到着分布に関して，一定の間隔で搬送系に到着すると仮定し，また，各機械での処理時間は正規分布（または，指数分布）に従うと仮定する．実際には，処理時間は正規分布に従うことが経験的に知られている．それゆえ，指数分布に従うと仮定するのは現実的ではないが，理論的には興味深い結果を得ることができる．これは，指数分布の特徴をうまく利用して解析したからである．

また，計算機シミュレーションを利用して衝突確率を求め，解析的に求めた結果と一致することを確認した．さらに，バッファを持つときは衝突確率にどのような影響を及ぼすかを，計算機シミュレーションを行って調べ，バッファの良い効果をいくつか得ることができた．

最後に，製造の担当者を悩ませている他の問題についても触れる．搬送計画問題は，多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．我々は，この問題を決定問題として定式化して，さらに NP 完全であることを証明した．そして，ある制約の下では多項式時間で解けることを示した．ジャストインタイムスケジューリング問題は，与えられた納期ちょうどにジョブの処理を完了するようなスケジュールを求める問題である．我々は周期的なタイムスロットという現実的な仮定の下で，この問題を取り扱う．そして，多項式時間で計算可能な任意の関数 $\alpha(n)$ に対して，P=NP でない限り，この問題を $\alpha(n)$ 以内の近似比で近似することは不可能である，という結果を得た．本研究では，機械数が 1 の場合に，高性能なヒューリスティックアルゴリズムを開発した．また，このアルゴリズムは妥当な制約の下で定数近似比を持つ近似アルゴリズムであることも証明した．

目 次

第 1章

まえがき

近年，半導体や液晶への需要が増しており，今まで以上に半導体や液晶を効率よく製造することが望まれる．そのためには，製造システムに対する適切な数学的モデルを構築することが重要である．本論文では，製造システムの中でも特に，全自動制御のシステムをモデル化する．このシステムでは，オブジェクト（ジョブ）が入口から投入されて，出口へと向かって流れていくのだが，その間にたくさんの機械で加工処理が行われる．各機械で処理が完了すると，次の機械へと自動的にオブジェクトが搬送される．

機械の接続関係には，かなり複雑なネットワーク型のシステムが存在するが，本論文では全ての機械が入口から出口まで直列につながっている直列型（タンデム型）を扱う．その理由は，一般のネットワーク型に対するモデルを構築し，衝突確率のような評価値を理論的に解析するのはかなり困難であると考えているからである．本論文で示すように，単純な直列型モデルでさえその振る舞いを解析するのは容易ではない．図 1.1 に直列型の搬送系の例を示す．

たくさんのオブジェクトが搬送系上を流れていくことになるが，ここで扱うオ

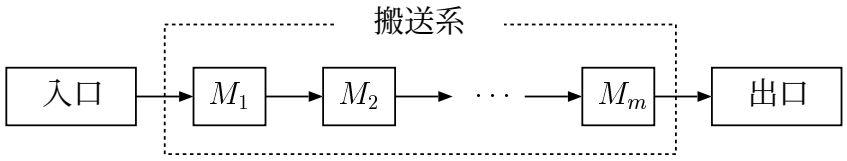


図 1.1: m 台の機械から成る直列型の搬送系.

ブジェクトは全て同一のものである．それゆえ理想的には，機械 M_j での処理時間は，任意のオブジェクトに対して同一になる．ときどき，異なるオブジェクトを扱う搬送系も存在するようである．この場合は，機械 M_j での処理時間がオブジェクト毎に異なる．しかし，ほとんどの直列型の搬送系では，同一のオブジェクトを扱う．

図 1.1 において，入口から投入されたオブジェクトは，最初に機械 M_1 へと運ばれる．オブジェクトが M_1 に到着すると，M_1 はすぐに処理を始める．M_1 が処理を完了するとすぐに，二番目の機械 M_2 へと運ばれる．このとき，もし M_2 が利用可能（空である）ならば，M_2 はすぐに運ばれてきたオブジェクトの処理を開始する．一方，もし M_2 がまだ他のオブジェクトを処理している最中であれば，M_2 に運ばれきたオブジェクトと M_2 で処理中のオブジェクトとがぶつかってしまう．このような状態をオブジェクト間の衝突と呼ぶ．

ところで，我々が対称としている半導体や液晶の製造では，入口から原材料となるガラス基板が投入される．投入後は一連の加工処理を行い，搬送系を退出してはじめて半導体や液晶が出来上がる．ガラス基板のサイズは，技術の進歩とともに大きくなる傾向がある．また，ガラス基板は 1 枚当りの価格が数十万円もする．さらに，ガラス基板は壊れやすい性質を持っている．そのため，できるだけ製造の途中で壊さないようにしなくてはならない．我々の直列型のモデルでは，オブジェクトは順番に $M_1, M_2, \ldots, M_m$ で処理されて，最後にその搬送系から退出する．ただし，退出するまでにできるだけオブジェクト間の衝突を避けるべきである．

たいていの現実に存在する半導体や液晶の製造装置では，搬送系へのオブジェクトの投入間隔は一定である．この一定の時間間隔のことを “タクトタイム” あるいは単に “タクト” と呼ぶ．ここでの目的は，全てのオブジェクトが衝突しないという条件の下で，最適な（最小の）タクトを求めることである．

もし前もって各機械での処理時間を全て知っていれば，最適なタクトを容易に求めることができる（詳細は次の章を参照のこと）．しかし，現実には最適なタクトを求めるのは困難であり，その理由を以下に述べる．先に述べたように同一のオブジェクトが大量に搬送系を流れていく．そのため，理想的には機械 M_j での処理時間は一定である．この場合には，次の章で述べるように，最適なタクトタイムを容易に求めることができる．しかし，実際には機械 M_j での処理時間は一定で

はない．実は，周りの環境（そのときの温度や，処理に使われる液体薬品の使用時間など）に依存して，そのつど機械 M_j での処理時間は変化する．この問題を解決するには，周りの環境の影響を受けない高性能な処理機械を利用すればよいが，そのような処理機械は今のところ現実的ではない．

そこで，処理時間に関して，確率モデルを導入する．幸いなことに，処理時間が正規分布に従うことが経験的に知られている．ただし，処理時間が正規分布や指数分布に従うとき，理論的には全く衝突無しに全てのジョブの処理を完了することを保証するのは不可能である（ただし，とても単純で，非現実的なやり方を除く）．

そのため，ここでの妥当な目的は，あらかじめ定められた衝突確率を満たすように，全てのジョブの処理を完了するという条件の下で，最小のタクトを求めることである．もっと正確に言えば，処理時間が正規分布（あるいは，指数分布）に従うと仮定して，ジョブ間の衝突確率を解析することである．

上述の問題は，典型的な機械スケジューリング問題と関連がある．機械スケジューリング問題とは，各機械で処理されるジョブの最適な処理順序を決定する問題である（例えば，[18] を参照）．また，機械スケジューリング問題の中でも特に，ブロッキング付きの待ち無し製造システム（no-wait production system with blocking）に関する多くの研究成果もある（広範囲な調査を行った [43] を参照）．このようなスケジューリング問題は，多品種製造システムを対象にしている．すなわち，全てのジョブが異なる，というのが前提条件である．この前提条件の下で，我々が取り組んだ研究を第 5 章で述べる．一方，第 2 章から第 4 章までは同一の製品（例えば，半導体や液晶など）を大量に製造する大量製造システムを扱う．それゆえ，各機械で処理されるジョブの最適な処理順序を求めるのは意味がない．

第 4.4 章ではバッファの効果について説明する．バッファは衝突を避けるために，各機械に用意される．理論的には，(1) 各機械が無限個のバッファを持っており，(2) 最初の機械 M_1 への到着間隔が指数分布（ポアソン到着）であり，(3) 各機械での処理時間が指数分布に従う，という条件が全て成り立つならば，そのようなシステムの振る舞いを解析することができる．実際，系人数の分布，系平均人数，系平均待ち時間などの解析結果が知られている [10]．これらの結果は，1 機械だけから成る待ち行列系で知られている結果（例えば [14] を参照）を利用することで

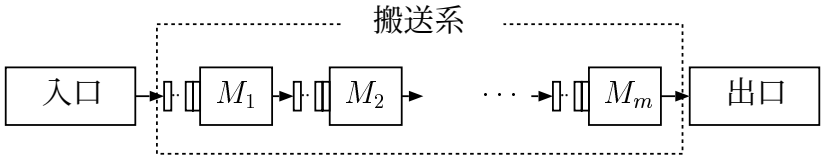


図 1.2: バッファ付きの m 台の機械から成る直列型の搬送系.

全て得ることができる．この場合，M_1 への到着間隔が指数分布に従い，M_1 での処理時間も指数分布に従うので，ここからの出力分布は入力と全く同じ指数分布に従う．そしてこれが M_2 への到着分布になるので，M_2 からの出力分布も同じ指数分布に従うという具合に，すべての機械からの出力分布が同一の指数分布に従う．従ってシステムが m 個の直列型で構成されていても，各機械を 1 つずつ切り離して考えることができるので，1 機械から成る待ち行列系の結果を使うだけで上記の結果がすべて得られる．

また，モデルを拡張すると解析するのが困難になる．例えば，機械のバッファの個数に制限のある直列型の解析はうまくできていない．バッファに制限のある直列型の解析は，一般に面倒な計算を伴うのが普通であって，まだあまり多くの結果は知られていない．

バッファ付きの m 機械から成る直列型の搬送系を図 1.2 に示す．このモデルでは，オブジェクトは最初に M_1 で処理をされて，その処理が終わると，次の機械 M_2 へと運ばれる．そのときもし M_2 が他のオブジェクトの処理を行っていれば，運ばれてきたオブジェクトは M_2 のバッファで待機することになる．M_2 が利用可能になると，自動的にバッファからオブジェクトを取り出し処理を開始する．一般に，M_j が複数のバッファを持っていれば，それらのバッファに複数のオブジェクトが待機する場合もある．その場合には M_j のバッファに待機した順番にオブジェクトを取り出す．このような動作が全て自動で行われる．

もし各機械のバッファの個数が無限に準備されているならば，それらのバッファを使ってオブジェクト同士の衝突を避けることができる．しかし，実際はバッファの個数は有限である．それゆえ，もしオブジェクトが M_i に到着したときに，M_i のバッファがすべて他のオブジェクトで占有されていると，衝突が起きる．この

ような状態のことを待ち行列理論の分野では，M_i がブロック（閉塞）されていると呼ぶ[10]．このブロック効果をもつようなモデルの解析は，かなり複雑になるため，あまり多くは知られていない．かなり厳しい条件の下で解析している場合がほとんどである．

関連研究として例えば，[15] では，2 つの機械のみから成る直列型で M_2 のバッファに制限のあるシステムを調べ，特にポアソン到着・一定サービスでの待ち時間分布や系人数の分布を求めている．[32] では，2 機械から成る待ち無しバッファ無し直列型待ち行列を扱っている．オブジェクトは入口にランダムにやってくるが，オブジェクト同士の衝突を避けるために，搬送系に投入するかどうかを決めるやり方を提案している．ただし，オブジェクトが入口に到着したときに，搬送系全体の情報（各機械での残りの処理時間など）を知ることができると仮定している．この情報を利用して，搬送系に投入するかどうかを決めるのである．我々のモデルでは，このような搬送系全体の情報を知ることはできない．[22] では，バッファ無しだが機械間の待ちを許す直列型待ち行列を研究した．[48] では，バッファ有りの 2 機械からなる直列型で，オブジェクトの損失率を最小化する問題を研究した．二番目の機械でのサービス率が与えられると，最初の機械での最適なサービス率と両方の機械に対する最適なバッファ割り当てを求める．

本論文では，オブジェクトの投入間隔が一定の値（タクト）である．言い換えると，搬送系への到着分布が単位分布[1] である．また，各機械での処理時間が正規分布（あるいは指数分布）に従う場合を考える．“タクトタイム” という用語は和製語で，もともとトヨタ生産方式[2] の一部として作られた（cf. [9], [24]）．これは，一つの製品を製造するのに要する時間を意味している．タクトという用語は，ドイツ語の “takt” [3] に由来するため，“Takt time” と独英混合となる．[7] では，説明無く “tact time” としているが，“tact” は，機転・感触・美的センスなどを意味し，タクトタイム本来の意味合いはない．これは日本独特の用法であり，英語では的確に示しにくいのが理由である．

メーカーなどの製造装置のユーザーが制御できることは，オブジェクトの投入間隔の設定である．ユーザーは各機械の処理時間を制御することができない．た

[1] 一定分布，あるいはレギュラー分布と呼ぶこともある．
[2] ジャストインタイム生産システムとしても知られている
[3] オーケストラの指揮者が振る指揮棒ないし拍子の意味

だし，多くの場合に処理時間が正規分布に従っていることが経験的に知られているようだ．

この論文のもう一つの目的は，我々のモデル上（すなわち，オンライン上）で決めたタクトと，各機械での処理時間を前もって全て知っている場合に決めたタクトを比較することである．各機械での処理時間の情報を使用しないで（分からないのでできない），その時点での動作（ここでは，前者のタクト）を決めなくてはいけないのがオンライン問題である．後者のタクトは現実的ではないが，最適なオフラインアルゴリズムで求めることができる．オンラインアルゴリズムと最適オフラインアルゴリズムとの間の性能を比較する競合比の解析 [13] が，オンライン問題に対して通常行われる．

ただし，最適なタクトに設定すると，衝突が起きそうになる危険な状態が多く発生するため，そのようなタクトは役に立たず，それを使うのはナンセンスである．理論的に衝突を起こらないことを保証するためには，次のようなナンセンスで陳腐なアルゴリズムを使わざるを得ない．すなわち，投入したオブジェクトが搬送系から退出してから，次のオブジェクトを投入する．しかし，このようなアルゴリズムを実現するのでさえ，オブジェクトが搬送系から退出したことを探知し，その処理終了の信号を入口へと送るセンサーが必要になる．我々のモデルにはそのようなセンサーはないと仮定している．

本論文では，到着分布が単位分布で，処理時間が正規分布（あるいは，指数分布）に従うとき，適切なタクトを求める方法を提案する．衝突確率が与えられたときに，その確率以下になるような最適な（最小の）タクトを，明示的に式で示すのはかなり困難である．しかし，高々3 台の機械から成る直列型搬送系で，処理時間が正規分布に従う場合は，二分探索法を利用して最適なタクトを求めることができる．一般に搬送系が m 台の機械から成る直列型である場合には，二分探索法の過程で m 重積分をする必要ある．しかし，$m > 5$ になると計算誤差のせいで正しい積分値を計算するのが困難になり，二分探索法をうまく実行することができない．実用的な用途では，タクトに関して厳密な値を必要としないので，擬似二分探索法（詳しくは，第 3.3.1 章を参照）で十分であると考えている．

一方，処理時間が指数分布に従う場合は事情が違ってくる．搬送系が 1 台の機械のみからなる場合，最適なタクトを明示的に式で表すことができる．一般に m

台から搬送系が構成される場合は，最適なタクトに対する明示的な式を与えるのは困難になるが，数値計算的には単に二分探索法を利用して求めることができる．その理由は，正規分布の場合と違って，衝突確率の式に積分を含んでおらず，容易に値を求めることができるからである．

以下では，本論文は次のように構成されている．第 2 章では，ジョブ同士が衝突しないための必要十分条件を与える．また，処理時間を決定的と仮定して，全体の処理時間を最小化する問題を考える．この章の最後では，一般の製造装置の表記法について述べる．第 3 章では，直列型の搬送系に限定して，ジョブ同士の衝突確率を解析的に導出する．ここでの議論の展開は，最も簡単な 1 機械からなる搬送系の場合からはじめて，2 機械，さらには m 機械へと拡張する．また，処理時間が正規分布に従う場合と指数分布に従う場合とを分けて解析する．第 4 章では，計算機シミュレーションの結果を述べる．結果の大部分は直列型搬送系での衝突確率であり，少しだけだが，2 機械からなる並列モデルでのシミュレーション結果も示す．また，各機械がバッファを持つときのシミュレーション結果も示す．第 5 章で，製造に関連した他のスケジューリング問題に関して，得られた結果を述べる．この章では，処理時間は決定的である．最後に第 6 章で，本論文のまとめと今後の課題について述べる．

第 2章

衝突条件，及び最適なオフラインアルゴリズム

本章から第 4.3 章までの間，図 1.1 に示したバッファ無し直列型搬送系を議論する．多くのジョブがこの搬送系に投入されて，処理をされた後に退出していく．ジョブはタクトタイムの間隔で搬送系に投入される．限られた時間で多くのジョブを処理するには，できるだけタクトタイムを短くすればよいが，短くした分だけ搬送系に存在するジョブの個数も多くなり，それゆえジョブ同士の衝突が起こりやすくなる．

本章では，どのような場合に衝突が起きるのかを考えて，衝突が生じるための必要十分条件を与える．この衝突に関する条件は，後の章で何度も用いられる．また，各機械での処理時間がすべて与えられたとき，最適スケジュールを計算するアルゴリズムを構築する．ここでは，あまり効率のよくないアルゴリズムから，非常に効率良いのアルゴリズムまで述べる．いろいろな手法で解くことができるため，理論的には興味深い．

簡単のために，機械間の搬送時間，入口から機械までの搬送時間及び機械から出口までの搬送時間が無い（すなわち，搬送時間が 0 である）と仮定する．実際は，このような搬送時間を無視することはできないが，処理時間の中に搬送時間を含んでいるとみなせば，ここでの内容をそのまま適用することが出来る．以下では次の表記が使われる．

- $M = \{M_1, M_2, \ldots, M_m\}$: m 台の機械からなる集合.

- $J = \{J_1, J_2, \ldots, J_n\}$: 処理される n 個のジョブからなる集合.
- $t_i^{(j)}$ (> 0): J_i の M_j での処理時間.
- T_{tact} (> 0): タクトタイム.

2.1 衝突条件

ジョブ間の衝突が起きないための必要十分条件を考える．仮にジョブの個数が一つならば（すなわち，$n = 1$），衝突が起きないのは明らかである．それゆえ，衝突に関して議論するときは $n \geq 2$ であると仮定する．まずは，1 台の機械のみからなる（すなわち，$m = 1$）搬送系を考える．ジョブ J_i $(2 \leq i \leq n)$ が入口から投入されたとき，一つ前に投入されたジョブ J_{i-1} が M_1 で処理中であれば衝突が起きる．

補題 1. $n \geq 2$ に対して，ジョブ間の衝突が起きないための必要十分条件は，すべての i $(1 \leq i \leq n-1)$ に対して $t_i^{(1)} \leq T_{tact}$ が成立することである．

証明. 全ての i $(1 \leq i \leq n-1)$ に対して，$t_i^{(1)} \leq T_{\mathrm{tact}}$ であれば，衝突が起こらないのは明らかである．また，$t_i > T_{\mathrm{tact}}$ を満たす i $(1 \leq i \leq n-1)$ に対して，ジョブ J_i と J_{i+1} が衝突する． (証明終)

補題 1 より，M_1 での J_n の処理時間（すなわち，$t_n^{(1)}$）は衝突に影響を与えない．$t_n^{(1)}$ がどんなに大きな（あるいは，小さい）値であっても，衝突に関して影響を与えない．

次に，2 台の機械からなるある搬送系ではどうだろう．あるジョブ J_i $(2 \leq i \leq n)$ が入口で投入されたとき，M_1 が一つ前に投入されたジョブ J_{i-1} を処理中であれば衝突が生じる．そうでなければ，M_1 は運ばれてきたジョブ J_i の処理を開始し，処理を終了するとすぐに，ジョブ J_i は M_2 へと運ばれる．このとき，もし M_2 が他のジョブ（すなわち，J_{i-1}）を処理中であれば，衝突が起きる．より正確に言えば，M_2 で衝突が起きるための必要十分条件は，M_2 が J_i の処理を完了するよりも先に，M_1 が J_{i+1} の処理を完了するような i $(1 \leq i \leq n-1)$ が存在することである．

補題 2. $n \geq 2$ に対して，ジョブ間の衝突が起きないための必要十分条件は，すべての i $(1 \leq i \leq n-1)$ に対して $t_i^{(1)} \leq T_{tact}$ かつ $t_i^{(1)} + t_i^{(2)} \leq T_{tact} + t_{i+1}^{(1)}$ が成立することである.

証明. 2 機械モデルの M_1 は 1 機械モデルの M_1 と同じ振舞いをする．それゆえ，2 機械モデルの M_1 で衝突が起きないための必要十分条件は，補題 1 よりすべての i $(1 \leq i \leq n-1)$ に対して $t_i^{(1)} \leq T_{\mathrm{tact}}$ が成立することである．

次に，M_2 で衝突が起きないための必要十分条件は，すべての i $(1 \leq i \leq n-1)$ に対して $t_i^{(1)} + t_i^{(2)} \leq T_{\mathrm{tact}} + t_{i+1}^{(1)}$ が成立することであることを証明しよう．最初の機械 M_1 が最初のジョブ J_1 の処理を開始する時刻を 0 とする．すると，M_2 がジョブ J_i $(1 \leq i \leq n-1)$ の処理を完了する時刻は

$$(i-1)\,T_{\mathrm{tact}} + t_i^{(1)} + t_i^{(2)}$$

であり，M_1 がジョブ J_{i+1} の処理を完了する時刻は

$$i\,T_{\mathrm{tact}} + t_{i+1}^{(1)}$$

である．よって，M_2 が J_i の処理を完了した後に M_1 が J_{i+1} の処理を完了するならば，すなわち式で書くと，

$$(i-1)\,T_{\mathrm{tact}} + t_i^{(1)} + t_i^{(2)} \leq i\,T_{\mathrm{tact}} + t_{i+1}^{(1)}$$

となり，この式を変形して

$$t_i^{(1)} + t_i^{(2)} \leq T_{\mathrm{tact}} + t_{i+1}^{(1)} \tag{2.1}$$

が成り立つならば，ジョブ J_i と J_{i+1} は M_2 で衝突が起きない．そして，全ての i $(1 \leq i \leq n-1)$ に対して，式 (2.1) が成り立つならばどのジョブも同士も M_2 で衝突しない．また，$t_i^{(1)} + t_i^{(2)} > T_{\mathrm{tact}} + t_{i+1}^{(1)}$ を満たす i $(1 \leq i \leq n-1)$ が存在すれば，$(i-1)\,T_{\mathrm{tact}} + t_i^{(1)} + t_i^{(2)} > i\,T_{\mathrm{tact}} + t_{i+1}^{(1)}$ を満たす i $(1 \leq i \leq n-1)$ に対して，M_2 がジョブ J_i の処理を完了するよりも先に M_1 がジョブ J_{i+1} の処理を完了する．よって，ジョブ J_i と J_{i+1} が M_2 で衝突する． (証明終)

補題 2 より，M_2 での J_n の処理時間（すなわち，$t_n^{(2)}$）は衝突に影響を与えない.

最後に一般の m 機械からなる搬送系を考えよう．あるジョブ J_i $(2 \leq i \leq n)$ が入口から投入されたとき，一つ前に投入されたジョブ J_{i-1} が M_1 で処理中であれば，M_1 で衝突が生じる．そうでなければ，M_1 は運ばれてきたジョブ J_i の処理をすぐに開始する．一般にすべての j $(2 \leq j \leq m)$ に対して，M_{j-1} がジョブの処理を完了するとすぐに，そのジョブは M_j に運ばれる．このとき，もし M_j が他のジョブを処理中であれば，M_j で衝突が生じる．より正確に述べると，M_j $(2 \leq j \leq m)$ で衝突が起きるための必要十分条件は，M_j が J_i の処理を完了するよりも先に，M_{j-1} が J_{i+1} の処理を完了するような i, j $(1 \leq i \leq n-1, 2 \leq j \leq m)$ が存在することである．衝突に関して，補題 1 と補題 2 の拡張として，次の補題 3 が得られる．

補題 3. $n \geq 2$ に対して，ジョブ間の衝突が起きないための必要十分条件は，すべての i,j $(1 \leq i \leq n-1, 1 \leq j \leq m)$ に対して

$$\sum_{k=1}^{j} t_i^{(k)} \leq T_{tact} + \sum_{k=1}^{j-1} t_{i+1}^{(k)}$$

が成立することである．

証明. 各機械 M_j $(\in M)$ で衝突が起こらないための必要十分条件が，全ての i $(1 \leq i \leq n-1)$ に対して

$$\sum_{k=1}^{j} t_i^{(k)} \leq T_{\mathrm{tact}} + \sum_{k=1}^{j-1} t_{i+1}^{(k)}$$

が成立することを示せばよい．m に関する帰納法で証明しよう．$m = 1$ のときは，補題 1 から主張は正しい．m より小さい m' で主張が正しいと仮定すると，結局は機械 M_m で衝突が起きないための必要十分条件が，すべての i $(1 \leq i \leq n-1)$ に対して

$$\sum_{k=1}^{m} t_i^{(k)} \leq T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)}$$

が成立することを証明すればよい．M_m がジョブ J_i $(1 \leq i \leq n-1)$ の処理を完了する時刻は

$$(i-1)\, T_{\mathrm{tact}} + \sum_{k=1}^{m} t_i^{(k)}$$

であり，M_{m-1} がジョブ J_{i+1} の処理を完了する時刻は

$$i\, T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)}$$

である．よって，M_m が J_i の処理を完了した後に M_{m-1} が J_{i+1} の処理を完了するならば，すなわち式で書くと

$$(i-1)\,T_{\mathrm{tact}} + \sum_{k=1}^{m} t_i^{(k)} \leq i\,T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)}$$

となり，この式を変形して

$$\sum_{k=1}^{m} t_i^{(k)} \leq T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)} \tag{2.2}$$

が成り立つならば，ジョブ J_i と J_{i+1} は M_m で衝突が起きない．そして，すべての i $(1 \leq i \leq n-1)$ に対して，式 (2.2) が成り立つならばどのジョブも同士も M_m で衝突しない．また，$\sum_{k=1}^{m} t_i^{(k)} > T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)}$ を満たす i $(1 \leq i \leq n-1)$ が存在すれば，$(i-1)\,T_{\mathrm{tact}} + \sum_{k=1}^{m} t_i^{(k)} > i\,T_{\mathrm{tact}} + \sum_{k=1}^{m-1} t_{i+1}^{(k)}$ を満たす i $(1 \leq i \leq n-1)$ に対して，M_m がジョブ J_i の処理を完了するよりも先に M_{m-1} がジョブ J_{i+1} の処理を完了する．よって，ジョブ J_i と J_{i+1} が M_m で衝突する． (証明終)

M_m で最後に処理される J_n の処理時間（すなわち，$t_n^{(m)}$）は衝突に影響を与えない．この事実は補題 3 に含まれている．

2.2 最適なオフラインアルゴリズム

すべての処理時間を前もって知っている（言い換えると，すべての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ が入力として与えられる）とき，次の問題を考えよう．

問題. ジョブ間の衝突が無いように全体の処理時間（すなわち，メイクスパン）を最小化せよ．

処理されるジョブは同じ種類のものなので，理想的には，各 $1 \leq j \leq m$ に対して，

$$t^{(j)} = t_1^{(j)} = t_2^{(j)} = \cdots = t_n^{(j)}$$

が成り立つ．このように，機械 M_j での処理時間を何番目のジョブかに関係なくある一定値 $t^{(j)}$ とみなすことができる．この場合は，最も大きな処理時間にタクトタイムを設定すればよい．それよりも大きな値に設定すれば，余分な遊び時間が

生じ，それよりも小さな値に設定すれば，必ず衝突が生じる．ゆえに，最適なタクトタイムを T^*_{tact} とすると，

$$T^*_{\mathrm{tact}} = \max_{1 \le j \le m} \{t^{(j)}\}.$$

である．さらに，このときの全体の処理時間は

$$(n-1)\, T^*_{\mathrm{tact}} + \sum_{j=1}^{m} t_n^{(j)}$$

となる．このようにして，理想的な処理機械を使用する場合には，最適なタクトタイムを容易に求めることができるが，そのような処理機械は現実的ではない．通常は異なる i, i' に対して，$t_i^{(j)}$ と $t_{i'}^{(j)}$ の値は異なる．そのような場合の最適タクトタイムの求め方を以下では考える．

全ての処理時間が前もって与えられたとき，後で述べるようないくつかのアルゴリズムにより最適なタクトタイムを求めることができる．これらのアルゴリズムは最適なオフラインアルゴリズムである．

ところで，各ジョブは搬送系に一定の時間周期（タクトタイム）で投入されるが，できるだけメイクスパンを短くするには，一定間隔でジョブを投入するのを止めて（タクト方式を止めて），投入間隔を可変にするほうがよい．そこでしばらくの間，投入間隔を可変として，上で述べた問題を考える．後で述べるように，投入間隔が可変である場合に考えたアルゴリズムを利用して，タクト方式の場合も容易に解くことができる．

以下では，M_1 が J_1 の処理を開始する時刻を 0，M_j が J_i の処理を開始する時刻を $x_i^{(j)}$ とし，処理を完了する時刻を $x_i^{(j+1)}$ とする．M_j が J_i の処理を完了する時刻 $x_i^{(j+1)}$ は，M_{j+1} が J_i の処理を開始する時刻に等しいことに注意しよう．なぜなら，M_j から M_{j+1} への搬送時間を無視している（0 とみなしている）からである．

2.2.1 線形計画法による解法

VLSI のレイアウトを 2 次元のコンパクション問題を解くことによって求めることができる [40]．具体的には，一次元のコンパクションアルゴリズムを 2 回使って求めている．[40] では，1 次元コンパクション問題の定式化も行われている．VLSI

のコンパクション手法では，レイアウト要素の座標間の制約を表現した制約グラフ [42] に基づいて，処理を行う．

一方，我々の問題で，全ての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ が前もって与えられているならば，矛盾の無い（衝突の無い）スケジュールを求めるのは容易である．ここで，この矛盾の無いスケジュールを VLSI のレイアウトに対応させることができる．そのため，時間軸に関して一次元コンパクション問題を解けばよい．以上のことから，以下に述べる線形計画問題を解くことによって，最小完了時間を求めることが出来る．

まずは，2 機械モデルで考える[1] ．2 機械モデルの場合は最小完了時刻は

$$t_1^{(1)} + \sum_{i=2}^{n} \max\{t_i^{(1)}, t_{i-1}^{(2)}\} + t_n^{(2)}$$

であることが知られている [21]．実は [21] では，搬送時間や投入に要する時間等も考慮に入れたより複雑なシステムを解析している．この解析結果を利用して，導出したのが上の式である．

図 2.1 には，ジョブ J_i が M_1 で処理を開始する時刻 $x_i^{(1)}$，M_1 で処理が終わる時刻 ($=M_2$ で処理を開始する時刻) $x_i^{(2)}$，M_2 で処理を完了する時刻 $x_i^{(3)}$ を使って，J_i のスケジュールを図示した．待ち無しの制約（M_1 で処理を終了後，すぐに M_2 で処理がはじまること）から次の式が成り立たなければならない．すべての i $(1 \leq i \leq n)$ に対して

$$x_i^{(1)} + t_i^{(1)} = x_i^{(2)},$$
$$x_i^{(2)} + t_i^{(2)} = x_i^{(3)}.$$

また図 2.2 に示したように，ジョブ J_i と J_{i+1} が M_1, M_2 で衝突しないための条件はそれぞれ次のようになる．全ての i $(1 \leq i \leq n-1)$ に対して

$$x_i^{(2)} \leq x_{i+1}^{(1)},$$
$$x_i^{(3)} \leq x_{i+1}^{(2)},$$

が成立することである．それゆえ，ジョブの個数が例えば 4 のとき，以下の線形

[1] 1 機械モデルでは明らかに最小完了時刻は $\sum_{i=1}^{n} t_i^{(1)}$ であり，議論するまでもない．

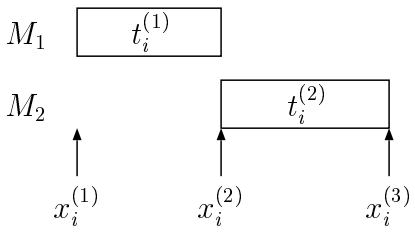


図 2.1: ジョブ J_i のスケジュールを図で示したもの

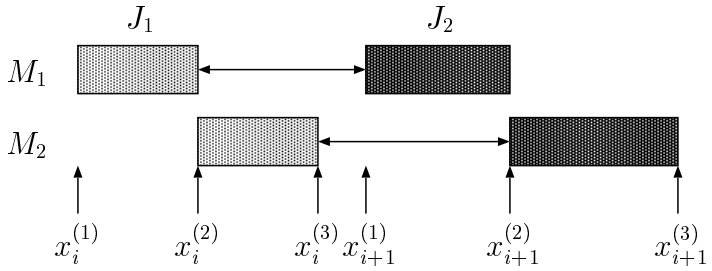


図 2.2: 制約条件

計画問題を解けばよい.

$$
\begin{aligned}
\text{最小化} \quad & x_4^{(3)} \\
\text{条件} \quad & x_1^{(1)} + t_1^{(1)} = x_1^{(2)}, \qquad x_1^{(2)} + t_1^{(2)} = x_1^{(3)}, \\
& x_2^{(1)} + t_2^{(1)} = x_2^{(2)}, \qquad x_2^{(2)} + t_2^{(2)} = x_2^{(3)}, \\
& x_3^{(1)} + t_3^{(1)} = x_3^{(2)}, \qquad x_3^{(2)} + t_3^{(2)} = x_3^{(3)}, \\
& x_4^{(1)} + t_4^{(1)} = x_4^{(2)}, \qquad x_4^{(2)} + t_4^{(2)} = x_4^{(3)}, \\
& x_1^{(2)} \le x_2^{(1)}, \qquad x_2^{(2)} \le x_3^{(1)}, \qquad x_3^{(2)} \le x_4^{(1)}, \\
& x_1^{(3)} \le x_2^{(2)}, \qquad x_2^{(3)} \le x_3^{(2)}, \qquad x_3^{(3)} \le x_4^{(2)}, \\
& x_1^{(1)} \ge 0, \qquad x_1^{(2)} \ge 0, \qquad x_1^{(3)} \ge 0, \\
& x_2^{(1)} \ge 0, \qquad x_2^{(2)} \ge 0, \qquad x_2^{(3)} \ge 0, \\
& x_3^{(1)} \ge 0, \qquad x_3^{(2)} \ge 0, \qquad x_3^{(3)} \ge 0, \\
& x_4^{(1)} \ge 0, \qquad x_4^{(2)} \ge 0, \qquad x_4^{(3)} \ge 0.
\end{aligned}
$$

上の線形計画問題を解くことにより得られる最適解 $x_i^{(j)}$ $(1 \le i \le 4, 1 \le j \le 3)$ は, 最適スケジュールに対応する.

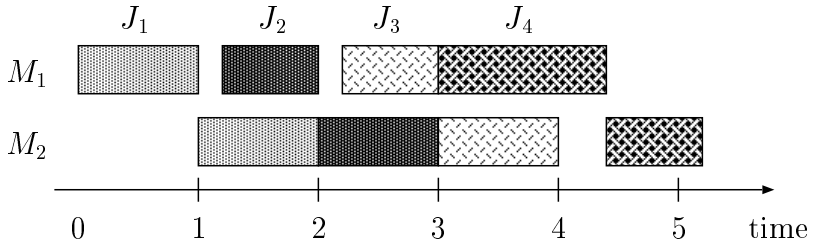


図 2.3: 最適スケジュール

例題 1. 以下の入力を考える.

$$t_1^{(1)} = 1, \quad t_1^{(2)} = 1, \quad t_2^{(1)} = 0.8, \quad t_2^{(2)} = 1,$$
$$t_3^{(1)} = 0.8, \quad t_3^{(2)} = 1, \quad t_4^{(1)} = 1.4, \quad t_4^{(2)} = 0.8.$$

上の入力に対して線形計画問題を解くと，最適解は下のようになる.

$$x_1^{(1)} = 0, \quad x_1^{(2)} = 1, \quad x_1^{(3)} = 2, \quad x_2^{(1)} = 1.2, \quad x_2^{(2)} = 2, \quad x_2^{(3)} = 3,$$
$$x_3^{(1)} = 2.2, \quad x_3^{(2)} = 3, \quad x_3^{(3)} = 4, \quad x_4^{(1)} = 3, \quad x_4^{(2)} = 4.4, \quad x_4^{(3)} = 5.2.$$

これらの $x_i^{(j)}$ $(1 \leq i \leq 4, 1 \leq j \leq 3)$ は最適スケジュールを意味しており，図 *2.3* に対応する.

一般にジョブ数 n の 2 機械モデルでは次の線形計画問題を解けばよい.

$$\begin{aligned}
\text{最小化} \quad & x_n^{(3)} \\
\text{条件} \quad & x_i^{(1)} + t_i^{(1)} = x_i^{(2)}, \quad x_i^{(2)} + t_i^{(2)} = x_i^{(3)} \quad (i = 1, 2, \ldots, n), \\
& x_i^{(2)} \leq x_{i+1}^{(1)}, \quad x_i^{(3)} \leq x_{i+1}^{(2)} \quad (i = 1, 2, \ldots, n-1), \\
& x_i^{(j)} \geq 0 \quad (i = 1, 2, \ldots, n, j = 1, 2, 3).
\end{aligned}$$

一般にジョブ数 n の m 機械モデルでは，2 機械モデルでの考え方を自然に拡張することで，次の線形計画問題を解けばよい.

$$\begin{aligned}
\text{最小化} \quad & x_n^{(m+1)} \\
\text{条件} \quad & x_i^{(j)} + t_i^{(j)} = x_i^{(j+1)} \quad (i = 1, 2, \ldots, n, j = 1, 2, \ldots, m), \\
& x_i^{(j+1)} \leq x_{i+1}^{(j)} \quad (i = 1, 2, \ldots, n-1, j = 1, 2, \ldots, m), \\
& x_i^{(j)} \geq 0 \quad (i = 1, 2, \ldots, n, j = 1, 2, \ldots, m+1).
\end{aligned}$$

線形計画問題を解くための様々なソルバーが存在する[2]．その中の一つで有名なフリーの lp_solve がある．現在，最新のバージョンは 5.5 であり，[23] から入手することができる．lp_solve を使う場合には，入力ファイルは lp 形式で書かなくてはならない．例えば，例題 1 の入力に対して，lp 形式で書き直したものを下に示す．

```
min: x_4_3;
c1: x_1_1 + 1 = x_1_2;
c2: x_1_2 + 1 = x_1_3;
c3: x_2_1 + 0.8 = x_2_2;
c4: x_2_2 + 1 = x_2_3;
c5: x_3_1 + 0.8 = x_3_2;
c6: x_3_2 + 1 = x_3_3;
c7: x_4_1 + 1.4 = x_4_2;
c8: x_4_2 + 0.8 = x_4_3;
c9: x_1_2 < x_2_1;
c10: x_2_2 < x_3_1;
c11: x_3_2 < x_4_1;
c12: x_1_3 < x_2_2;
c13: x_2_3 < x_3_2;
c14: x_3_3 < x_4_2;
```

lp 形式にはいくつかの約束がある．例えば，分の区切りには ; を打つ．変数は自動的に非負と仮定される．括弧 ( ) を使ってはいけない．c1: , c2: , $\cdots$ は制約式を分かりやすくするために書いたもので，省略してもよい．また，<, > をそれぞれ <=, >= と書いてもよい．上に示した lp 形式で書いた入力ファイル名を example.lp とする．lp_solve を実行するには，コマンドプロンプトを立ち上げ，

```
lp_solve < example.lp
```

と入力すると，以下のように出力される．

[2][25] では，線形計画問題に関連するソフトウェアだけでなく，様々な最適化ソフトウェアに関する情報を入手できる

```
Value of objective function: 5.2


Actual values of the variables:
x_4_3                          5.2
x_1_1                            0
x_1_2                            1
x_1_3                            2
x_2_1                          1.2
x_2_2                            2
x_2_3                            3
x_3_1                          2.2
x_3_2                            3
x_3_3                            4
x_4_1                            3
x_4_2                          4.4
```

これらの解は 例題 1 で与えたものと一致する.

2.2.2 ネットワーク理論による解法

これまで考えてきた 1 次元コンパクション問題は，ネットワーク理論を利用して解くこともでき (cf. [20])，線形計画法で求めるよりも高速に解くことが出来る. まずは，矛盾の無いスケジュールから制約グラフへと変換する．すなわち，次のような有向グラフ $G = (V, A)$ を構成する.

- $V = \{v_i^{(j)} | 1 \leq i \leq n, 1 \leq j \leq m+1\}$ であり，$mn + n$ 個の点から成る.
- A は以下に示す $3mn - m$ 個の枝から成る．ただし，枝に重みを割り当てる関数を w とする.
  - 重み $w(v_i^{(j)}, v_i^{(j+1)}) = t_i^{(j)}$ を付した $(v_i^{(j)}, v_i^{(j+1)})$ $(1 \leq i \leq n, 1 \leq j \leq m)$
  - 重み $w(v_i^{(j+1)}, v_i^{(j)}) = -t_i^{(j)}$ を付した $(v_i^{(j+1)}, v_i^{(j)})$ $(1 \leq i \leq n, 1 \leq j \leq m)$

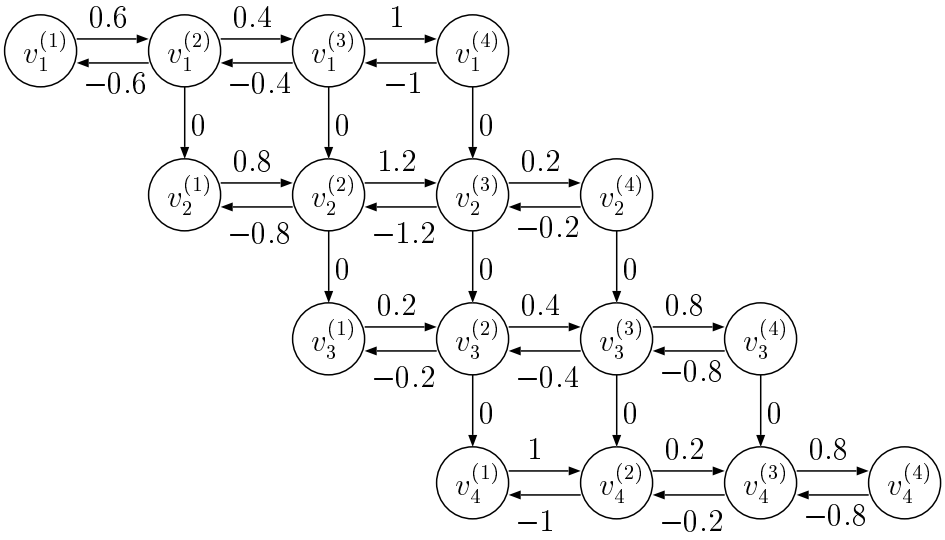


図 2.4: グラフ G の例

– 重み $w(v_i^{(j)}, v_{i+1}^{(j-1)}) = 0$ を付した $(v_i^{(j)}, v_{i+1}^{(j-1)})$ $(1 \leq i \leq n-1, 2 \leq j \leq m+1)$

ただし，$v_1^{(1)}$ を始点で，$v_n^{(m+1)}$ を終点とする．

例題 2. 以下の入力に対して，構成される有向グラフ G を図 2.4 に示す.

ジョブ数 $n = 4$, 機械数 $m = 3$,

$t_1^{(1)} = 0.6, \quad t_1^{(2)} = 0.4, \quad t_1^{(3)} = 1, \quad t_2^{(1)} = 0.8, \quad t_2^{(2)} = 1.2, \quad t_2^{(3)} = 0.2,$

$t_3^{(1)} = 0.2, \quad t_3^{(2)} = 0.4, \quad t_3^{(3)} = 0.8, \quad t_4^{(1)} = 1, \quad t_4^{(2)} = 0.2, \quad t_4^{(3)} = 0.8.$

始点，終点はそれぞれ $v_1^{(1)}, v_4^{(4)}$ である．

上のように構成した有向グラフ G に対して，最小完了時刻は始点から終点までの最長路の長さに対応する．そのため最長路問題を解けばよいが，一般の最長路問題は代表的な NP 困難問題の一つであり，効率的に解くアルゴリズムは知られていない [39]．しかし，上のように構成した有向グラフ G 上には正のサイクルが存在しないため，次のようにして効率的に解くことが出来る．G 上の全ての枝に対して，重みの符号を逆にする．すなわち，すべての枝 $(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) \in A$ に対して

$$\bar{w}(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) \equiv -w(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}).$$

このようにして得られた有向グラフを $\bar{G}$ とすると，$\bar{G}$ に対して最短路問題を解くことになる．$\bar{G}$ 上には，負のサイクルが存在しないため，Bellman-Ford 法を利用して，$O(|V||A|) = O(m^2n^2)$ で解くことが出来る（例えば，[49] などを参照）．さらに次に説明するようにやれば，ダイクストラ法 [19] を利用して解くことができる．このとき，データ管理に単純なデータ構造を使うと，計算時間は $O(|A| + |V|^2) = O(m^2n^2)$ であり，Bellman-Ford 法を利用したときと変わらない．しかし，データ管理にヒープを用いると $O(|A| \log |V|) = O(mn(\log m + \log n))$ の計算時間で解くことができる．また，Fredman と Tarjan [38] によるフィボナッチヒープという特殊なデータ構造を用いると，計算時間は $O(|A| + |V| \log |V|) = O(mn(\log m + \log n))$ となり，結局のところ単にヒープを用いるときと計算量は同じになる．

まず，衝突が起きないような初期スケジュールを求める．例えば，ジョブ J_i $(1 \leq i \leq n-1)$ が機械 M_m で終了した時点で，次のジョブ J_{i+1} の処理を M_1 で開始すれば，ジョブ J_i と J_{i+1} が衝突することは無い．このように初期スケジュールを決めたとき，M_j がジョブ J_i の処理を開始する時刻を $x_i^{(j)'}$ とし，M_m がジョブ J_i の処理を終了する時刻を $x_i^{(m+1)'}$ とする．また，有向グラフ G の全ての枝に対して，次のように枝の重みを付け替えた有向グラフを $\tilde{G}$ とする．すべての枝 $(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) \in A$ に対して

$$\begin{aligned}\tilde{w}(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) &\equiv -x_{i_1}^{(j_1)'} + \bar{w}(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) + x_{i_2}^{(j_2)'} \\ &= -x_{i_1}^{(j_1)'} - w(v_{i_1}^{(j_1)}, v_{i_2}^{(j_2)}) + x_{i_2}^{(j_2)'} \geq 0.\end{aligned}$$

このようにして得られた $\tilde{G}$ に対して，最短経路問題を解く（このテクニックは，[27] と [44] で開発された）．$\tilde{G}$ 上の全ての枝の重みが非負なので，ダイクストラ法を利用して求めることができ，得られた始点 $v_1^{(1)}$ から $v_i^{(j)}$ までの最短距離 $d_i^{(j)}$ の分だけ初期スケジュール $x_i^{(j)'}$ から短縮することが出来る．すなわち，下の式が成り立つ．すべての i,j $(1 \leq i \leq n, 1 \leq j \leq m+1)$ に対して

$$x_i^{(j)} = x_i^{(j)'} - d_i^{(j)}.$$

このようにして得られた $x_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m+1)$ は最適スケジュールに対応する．このとき，メイクスパンは $x_n^{(m+1)}$ である．

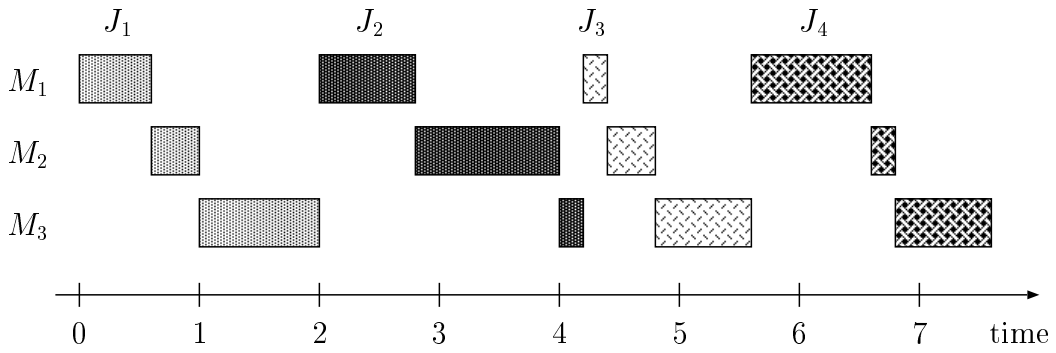


図 2.5: 初期スケジュールの例

例題 3. 以下の入力（例題 2 と同じ）を考える.

ジョブ数 $n = 4$, 機械数 $m = 3$,

$$t_1^{(1)} = 0.6, \quad t_1^{(2)} = 0.4, \quad t_1^{(3)} = 1, \quad t_2^{(1)} = 0.8, \quad t_2^{(2)} = 1.2, \quad t_2^{(3)} = 0.2,$$
$$t_3^{(1)} = 0.2, \quad t_3^{(2)} = 0.4, \quad t_3^{(3)} = 0.8, \quad t_4^{(1)} = 1, \quad t_4^{(2)} = 0.2, \quad t_4^{(3)} = 0.8.$$

上の入力に対して，初期スケジュールを以下に示す.

$$x_1^{(1)'} = 0, \quad x_1^{(2)'} = 0.6, \quad x_1^{(3)'} = 1, \quad x_1^{(4)'} = 2,$$
$$x_2^{(1)'} = 2, \quad x_2^{(2)'} = 2.8, \quad x_2^{(3)'} = 4, \quad x_2^{(4)'} = 4.2,$$
$$x_3^{(1)'} = 4.2, \quad x_3^{(2)'} = 4.4, \quad x_3^{(3)'} = 4.8, \quad x_3^{(4)'} = 5.6,$$
$$x_4^{(1)'} = 5.6, \quad x_4^{(2)'} = 6.6, \quad x_4^{(3)'} = 6.8, \quad x_4^{(4)'} = 7.6.$$

この初期スケジュールを図 2.5 に示す．初期スケジュール $x_i^{(j)'}$ を使って構成されたグラフ $\tilde{G}$ を図 2.6 に示す．$\tilde{G}$ において，始点 $v_1^{(1)}$ から各点までの最短距離 $d_i^{(j)}$ を下に示す.

$$d_1^{(1)} = 0, \quad d_1^{(2)} = 0, \quad d_1^{(3)} = 0, \quad d_1^{(4)} = 0,$$
$$d_2^{(1)} = 1.4, \quad d_2^{(2)} = 1.4, \quad d_2^{(3)} = 1.4, \quad d_2^{(4)} = 1.4,$$
$$d_3^{(1)} = 1.8, \quad d_3^{(2)} = 1.8, \quad d_3^{(3)} = 1.8, \quad d_3^{(4)} = 1.8,$$
$$d_4^{(1)} = 3, \quad d_4^{(2)} = 3, \quad d_4^{(3)} = 3, \quad d_4^{(4)} = 3.$$

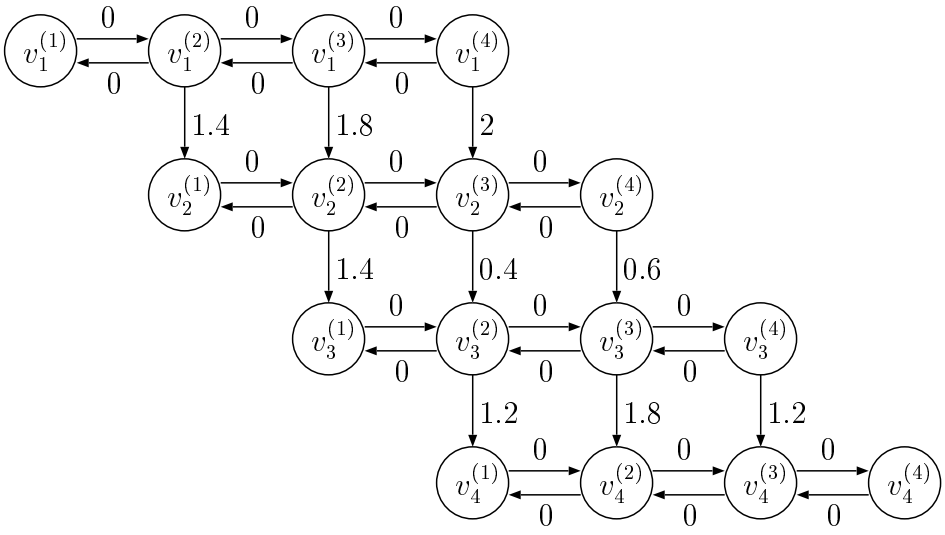


図 2.6: グラフ $\tilde{G}$ の例

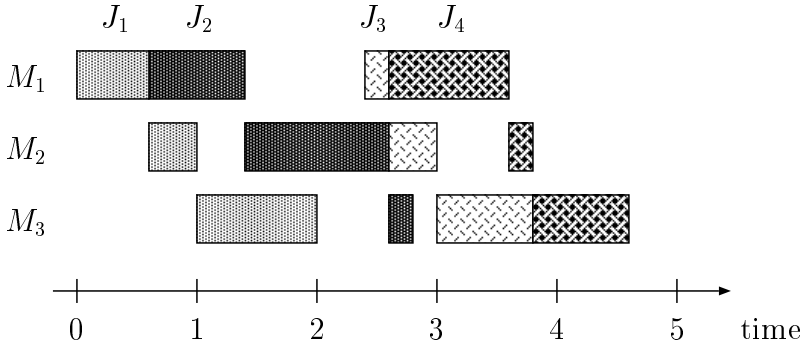


図 2.7: 最適スケジュール

従って，求めるべき最適スケジュール $x_i^{(j)} = x_i^{(j)\prime} - d_i^{(j)}$ は下のようになる．

$$
\begin{aligned}
x_1^{(1)} = 0, \quad x_1^{(2)} = 0.6, \quad x_1^{(3)} = 1, \quad x_1^{(4)} = 2, \\
x_2^{(1)} = 0.6, \quad x_2^{(2)} = 1.4, \quad x_2^{(3)} = 2.6, \quad x_2^{(4)} = 2.8, \\
x_3^{(1)} = 2.4, \quad x_3^{(2)} = 2.6, \quad x_3^{(3)} = 3, \quad x_3^{(4)} = 3.8, \\
x_4^{(1)} = 2.6, \quad x_4^{(2)} = 3.6, \quad x_4^{(3)} = 3.8, \quad x_4^{(4)} = 4.6.
\end{aligned}
$$

この最適スケジュールを図 2.7 に示す．

2.2.3 貪欲法による解法

上述の線形計画法とネットワークフローアルゴリズムを利用して最適スケジュールを求めるのを止めて，ここでは貪欲アルゴリズムを提案する．すべての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ が与えられたとき，ジョブ J_1 から J_n まで順番に，待ち無し制約の下で，できるだけ早く各ジョブの処理が完了するようにスケジュールを求めていく．以下に C 言語風に書いたコードを示す．

入力: n, m, 全ての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$.
出力: メイクスパン.

```
1.   // ジョブ J_1 のスケジュールを計算する.
2.   x_1^(1) = 0;
3.   for (j = 2; j ≤ m + 1; j++)
4.       x_1^(j) = x_1^(j−1) + t_1^(j−1);
5.   // ジョブ J_2, J_3, ..., J_n のスケジュールを計算する.
6.   for (i = 2; i ≤ n; i++) {
7.       for (j = 1; j ≤ m; j++) {
8.           x_i^(j) = x_{i−1}^(j+1);
9.           for (k = j + 1; k ≤ m + 1; k++)
10.              x_i^(k) = x_i^(k−1) + t_i^(k−1);
11.          for (k = j − 1; k ≥ 1; k−−)
12.              x_i^(k) = x_i^(k+1) − t_i^(k);
13.          flag = 0;
14.          for (k = 2; k ≤ m + 1; k++)
15.              if (x_{i−1}^(k) > x_i^(k−1)) flag = 1;
16.          if (flag == 0) break;    // 衝突のチェック
17.      }
18.  }
19.  return x_n^(m+1);
```

1–4 行目で J_1 のスケジュールを求める．J_1 の M_1 での処理開始時刻は 0 である．J_1 の M_j での処理開始時刻は，M_{j-1} での処理開始時刻に M_{j-1} での処理時間を加えたものなので，$x_1^{(j-1)} + t_i^{(j-1)}$ である．

残りの行で，J_2 から J_n までのスケジュールを求める．その際，下に示した最適スケジュールの性質を利用している．

最適スケジュールの性質

各 J_i $(2 \leq i \leq n)$ の最適スケジュールに関して，M_j が J_i の処理を開始する時刻に，M_j が J_{i-1} の処理を完了するような j $(1 \leq j \leq m)$ が存在する．

J_i の M_j での処理開始時刻 $x_i^{(j)}$ を J_{i-1} の M_j での処理完了時刻にあわせる．これは 8 行目に対応する．このように仮の $x_i^{(j)}$ を決めると，待ち無し制約から，他のすべての機械 M_k $(\in M \setminus \{M_j\})$ での処理開始 (終了) 時刻も求まる．これは，9–12 行目に対応する．次に，このようにして求めた J_i のスケジュールに矛盾がないか（すなわち，J_{i-1} と J_i が衝突していないか）をチェックする．これは，14–16 行目に対応する．

以上の処理をすべての i, j $(2 \leq i \leq n, 1 \leq j \leq m)$ に対して行う．それゆえ，3 重ループの構造になるので，計算時間は $O(m^2 n)$ となる．このようにアルゴリズムを設計すると，前述のネットワークフローアルゴリズムを利用した方が計算時間の面で優れている．実は上で示したコードは改善の余地があり，2 重ループの構造に書き直すことができる．

下に示した関数 Greedy Algorithm は 2 重ループの構造を持つ．このアルゴリズムも順番にジョブ $J_1, J_2, \ldots, J_n$ の処理ができるだけ早く完了するように各ジョブのスケジュールを計算する．

```
function Greedy Algorithm
入力: n, m, 全ての処理時間 t_i^(j) (1 ≤ i ≤ n, 1 ≤ j ≤ m).
出力: メイクスパン.
 1.   // ジョブ J_1 のスケジュールを計算する.
 2.   x_1^(1) = 0;
 3.   for (j = 2; j ≤ m + 1; j++)
 4.       x_1^(j) = x_1^(j-1) + t_1^(j-1);
 5.   // ジョブ J_2, J_3, ..., J_n のスケジュールを計算する.
 6.   for (i = 2; i ≤ n; i++) {
 7.       x_i^(1) = x_{i-1}^(m+1);
 8.       minsf = x_i^(1) − x_{i-1}^(2);
 9.       for (j = 2; j ≤ m; j++) {
10.           x_i^(j) = x_i^(j-1) + t_i^(j-1);
11.           temp = x_i^(j) − x_{i-1}^(j+1);
```

```
12.              if (temp < minsf) minsf = temp;
13.          }
14.          x_i^(m+1) = x_i^(m) + t_i^(m);
15.          for (j = 1; j ≤ m + 1; j++)
16.              x_i^(j) = x_i^(j) − minsf;    // x_i^(j) の値を更新する.
17.      }
18.      return x_n^(m+1);
```

1–4 行目で最初のジョブ J_1 のスケジュールを求める．残りの行は，順番に J_2, J_3, ..., J_n のスケジュールを求める．まず最初にジョブ間の衝突が無いという制約を満たすような J_i の初期スケジュールを求める．関数 Greedy Algorithm では，M_1 が J_i の処理を開始する時刻を，M_m が J_{i-1}（J_i の一つ前に投入されたジョブ）の処理を完了した時刻にあわせる．これは 7 行目に対応する．M_1 が J_i の処理を開始する時刻を決めると，待ち無し制約から，残りのすべての機械 M_j ($\in M \setminus \{M_1\}$) が J_i の処理を開始（完了）する時刻がすべて決まる．これは 9，10 行目に対応する．以上で，J_i の初期スケジュールが求まる．

J_i の初期スケジュールには，無駄な遊び時間（各機械での，J_{i-1} の処理完了時刻と J_i の処理開始時刻の差）が存在するので，できるだけこの遊び時間を短くしなければならない．J_i の初期スケジュールに存在する遊び時間のうち短縮可能な時間は，各機械の遊び時間の最小値（すなわち，16 行目の minsf [3] ）である．これよりも多くの時間を短縮すると，必ず衝突が生じる．これよりも少ない時間を短縮すると，無駄な遊び時間が存在する．各機械の遊び時間の最小値の計算は 8–13 行目で行う．

J_i の初期スケジュール $x_i^{(j)}$ $(1 \leq j \leq m+1)$ から，上述の minsf の分だけ引いたものが，J_i の最適スケジュールとなる．これは 15，16 行目に対応する．そして，関数 Greedy Algorithm は，18 行目でメイクスパンを返す．

このアルゴリズムは二重ループの構造を持つので，計算時間は明らかに $\Theta(mn)$ である．m と n が正の整数（$m = 1, n = 1$ のようなケースも含む）であれば，アルゴリズムは正しく動作する．さらに，全ての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ を入力するのに $\Theta(mn)$ 時間かかるので，計算時間に関して，このアルゴリズムは最適である．

[3] minsf は “minimum so far” （今までのところ最小）の略

補題 4. 関数 *Greedy Algorithm* は最小のメイクスパンを達成するスケジュール $\left(x_i^{(j)}\ (1 \leq i \leq n, 1 \leq j \leq m+1)\right)$ を計算する．

証明. 内側の最初の for ループ（9 行目のループ）に関して，インデックスの値が j のときに，ループが実行された後における minsf の値を $\mathrm{minsf}(j)$ とする．最初は，

$$\mathrm{minsf}(1) = x_i^{(1)} - x_{i-1}^{(2)}$$

である．この値が内側のループの中で更新されて，

$$\mathrm{minsf}(2) = \min\{\mathrm{minsf}(1), x_i^{(2)} - x_{i-1}^{(3)}\}$$

となる．この操作を繰り返して，一般には

$$\mathrm{minsf}(j) = \min\{\mathrm{minsf}(j-1), x_i^{(j)} - x_{i-1}^{(j+1)}\}$$

となる．変数 minsf に注目すると，上の再帰的な関係より，$\mathrm{minsf}(j)$ が $x_i^{(1)} - x_i^{(2)}$ から $x_i^{(j)} - x_{i-1}^{(j+1)}$ までの最小値を表していることがわかる．つまり，

$$\mathrm{minsf}(j) = \min\{x_i^{(1)} - x_{i-1}^{(2)}, x_i^{(2)} - x_{i-1}^{(3)}, \ldots, x_i^{(j)} - x_{i-1}^{(j+1)}\}$$

である．したがって，$j = m$ のときには，すべての機械を考慮して，ジョブ J_{i-1} と J_i 間の遊び時間の最小値が $\mathrm{minsf}(m)$ として求まっていることが分かる．

さて，J_1 のスケジュール $x_1^{(j)}\ (1 \leq j \leq m+1)$ が最適であるのは明らかである．J_2 のスケジュール $x_2^{(j)}\ (1 \leq j \leq m+1)$ を求めるときは，J_2 の初期スケジュールから $\mathrm{minsf}(m)$ の分だけずらすので，$x_2^{(j)}$ は最適スケジュールである．i より小さい値 i' に対して，J_1 から $J_{i'}$ が最適にスケジュールされていると仮定する．J_i のスケジュール $x_i^{(j)}\ (1 \leq j \leq m+1)$ を求めるときも，J_i の初期スケジュールから $\mathrm{minsf}(m)$ の分だけずらすので，$x_i^{(j)}$ は最適スケジュールである．よって，J_1 から J_n までのスケジュールは最適である． (証明終)

例題 4. 実際に関数 *Greedy Algorithm* を実装して，以下の入力で実行した．

$$\begin{aligned}
&n = 4, \quad m = 4,\\
&t_1^{(1)} = 3, \quad t_1^{(2)} = 4, \quad t_1^{(3)} = 2, \quad t_1^{(4)} = 5,\\
&t_2^{(1)} = 2, \quad t_2^{(2)} = 3, \quad t_2^{(3)} = 5, \quad t_2^{(4)} = 2,\\
&t_3^{(1)} = 6, \quad t_3^{(2)} = 2, \quad t_3^{(3)} = 2, \quad t_3^{(4)} = 4,\\
&t_4^{(1)} = 2, \quad t_4^{(2)} = 3, \quad t_4^{(3)} = 3, \quad t_4^{(4)} = 3.
\end{aligned}$$

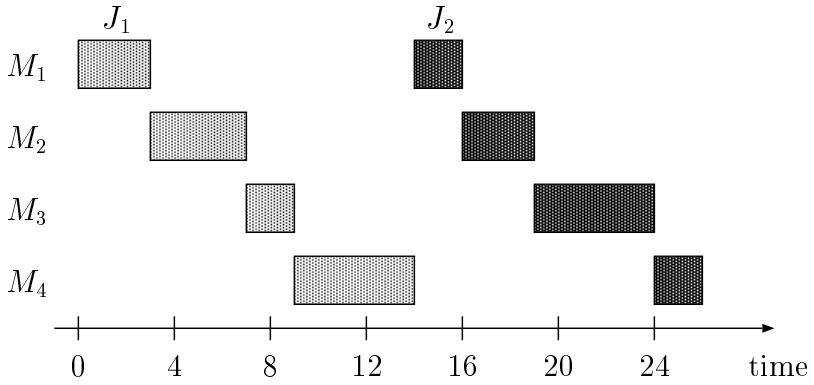


図 2.8: J_2 の初期スケジュール

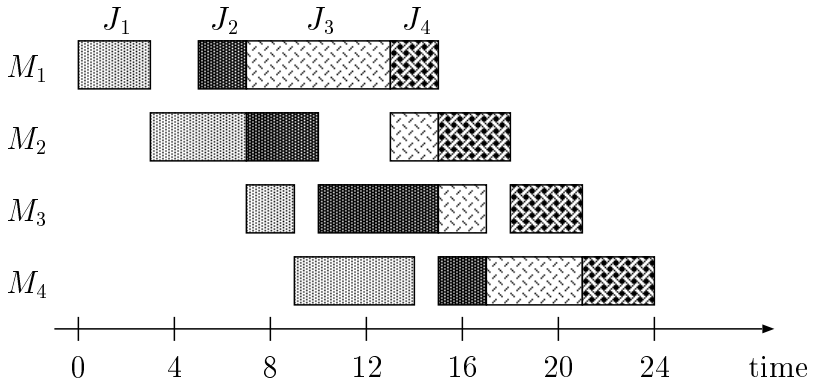


図 2.9: 最適スケジュール

J_2 の初期スケジュールを図 2.8 に示す．メイクスパン（すなわち，$x_4^{(5)}$）は，24 となる．また，関数実行後に求められた $x_i^{(j)}$ $(1 \leq i \leq 4, 1 \leq j \leq 5)$ を下に示す．

$$x_{11} = 0, \quad x_{12} = 3, \quad x_{13} = 7, \quad x_{14} = 9, \quad x_{15} = 14,$$
$$x_{21} = 5, \quad x_{22} = 7, \quad x_{23} = 10, \quad x_{24} = 15, \quad x_{25} = 17,$$
$$x_{31} = 7, \quad x_{32} = 13, \quad x_{33} = 15, \quad x_{34} = 17, \quad x_{35} = 21,$$
$$x_{41} = 13, \quad x_{42} = 15, \quad x_{43} = 18, \quad x_{44} = 21, \quad x_{45} = 24.$$

これらの結果は最適スケジュールを意味しており，図 2.9 に対応する．

次に，計算手間は同じだが，初期スケジュールの作り方が異なるアルゴリズムを示す．ジョブ J_1 のスケジュールの決め方は，前述の Greedy Algorithm と同じ

である．J_i $(2 \leq i \leq n)$ のスケジュールを決めるとき，M_1 が J_i の処理を開始する時刻を，M_1 が一つ前に投入されたジョブ J_{i-1} の処理を完了した時刻に合わせる．M_1 が J_i の処理を開始する時刻を決めると，待ち無しの制約から，他のすべての機械 M_j $(\in M \setminus \{M_1\})$ が J_i の処理を開始（完了）する時刻も決まる．このようにして決めたのが，J_i の初期スケジュールである．ただし，この J_i の初期スケジュールには，J_{i-1} と J_i の衝突が生じる可能性がある．

J_{i-1} と J_i 間に衝突が存在するとは，機械 M_j $(2 \leq j \leq m)$ がジョブ J_i の処理を開始する時刻が，M_j がジョブ J_{i-1} の処理を終了する時刻よりも先になっていることに対応する．そこで，衝突が存在する全ての機械 M_j において，それらの時刻の差の絶対値のうち，最大値の分だけ J_i の初期スケジュールをずらせばよい．それよりも多くずらすと，少なくともそのずらした分だけ無駄な遊び時間になる．このような考え方に基づいて書いたのが，関数 Greedy Algorithm 2 である．

function *Greedy Algorithm* 2
入力: n, m, 全ての処理時間 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$.
出力: メイクスパン.

1. // ジョブ J_1 のスケジュールを計算する.
2. $x_1^{(1)} = 0$;
3. **for** $(j = 2; j \leq m+1; j++)$
4. $\quad x_1^{(i)} = x_1^{(i-1)} + t_1^{(i-1)}$;
5. // ジョブ $J_2, J_3, \ldots, J_n$ のスケジュールを計算する.
6. **for** $(i = 2; i \leq n; i++)$ {
7. $\quad x_i^{(1)} = x_{i-1}^{(2)}$;
8. $\quad$ maxsf $= 0$;
9. $\quad$ **for** $(j = 2; j \leq m; j++)$ {
10. $\quad\quad x_i^{(j)} = x_i^{(j-1)} + t_i^{(j-1)}$;
11. $\quad\quad$ temp $= x_{i-1}^{(j+1)} - x_i^{(j)}$;
12. $\quad\quad$ **if** (maxsf $<$ temp) maxsf $=$ temp;
13. $\quad$ }
14. $\quad x_i^{(m+1)} = x_i^{(m)} + t_i^{(m)}$;
15. $\quad$ **for** $(j = 1; j \leq m+1; j++)$
16. $\quad\quad x_i^{(j)} = x_i^{(j)} +$ maxsf; $\quad$ // $x_i^{(j)}$ の値を更新する.
17. }

18. **return** $x_n^{(m+1)}$;

さて，扱う対象をタクト方式の搬送系に戻る．全ての処理時間が前もって与えられたとき，衝突が起こらないように，メイクスパンを最小にするような最適なタクトタイム $T^*_{\rm tact}$ を求める問題を考える．この問題は，先に記述した関数 Greedy Algorithm を使って容易に解くことができる．すなわち，関数 Greedy Algorithm の実行後，得られたスケジュールの投入間隔の最大値が，最適なタクトタイムである．それよりも小さな値をタクトタイムとして設定すると，衝突が生じる．それゆえ，$T^*_{\rm tact}$ は，

$$\max_{1\leq i\leq n-1}\{x_{i+1}^{(1)}-x_i^{(1)}\}$$

である．

次に，最適なオフラインアルゴリズムを使って求めた最適なタクト $T^*_{\rm tact}$ と，オンライン上で決めたタクト $T_{\rm tact}$ との比 $T_{\rm tact}/T^*_{\rm tact}$ の最大値（すなわち，競合比）を考える．競合比の概念は，衝突が存在しないという制約の下で定義されるものである．補題 1 より，M_1 で衝突なしのための必要十分条件は $t_i^{(1)} \leq T_{\rm tact}$ である．搬送系をオンライン上で考えるときは，処理時間 $t_i^{(j)}$ は正規分布（または，指数分布）に従うので，値としては無限に大きくなりうる．それゆえ，$T_{\rm tact}$ を無限大に設定しておかないと，衝突無しを保障できない．よって，$T^*_{\rm tact}$ に関係なく，$T_{\rm tact}=\infty$ としなければならない．

我々が対象としているタクトタイム方式の自動搬送システムでは，理論的に衝突無しを保障するためには，タクトタイムを無限大に設定しなくてはならない．

そこで，タクトタイム方式を止めて，投入間隔をいつでも変更することができる場合を考える．搬送系の最後の機械に終了センサーを用意する．終了センサーとは，処理が終了したことを認識する装置である．このセンサーを使えば，次のような簡単なオンラインアルゴリズム A を考えることができる．すなわち，ジョブが搬送系から退出したときに，次のジョブを搬送系へと投入する，というものである．入力を I としたとき，オンラインアルゴリズムを使ったときの全体の処理時間を $A(I)$ とする．また，最適オフラインアルゴリズムを使ったときの全体の処理時間を $OPT(I)$ とする．このとき，オンラインアルゴリズムの競合比は，あらゆる I に関して，$A(I)/OPT(I)$ の最大値で定義される．

補題 5. オンラインアルゴリズム A の競合比は $\min\{m, n\}$ である.

証明. 次の式が成り立つ.

$$A(I) = \sum_{i=1}^{n} \sum_{j=1}^{m} t_i^{(j)} \leq m \cdot \max_{1 \leq j \leq m} \left\{ t_1^{(j)} + t_2^{(j)} + \cdots + t_n^{(j)} \right\} \tag{2.3}$$

$$OPT(I) \geq \max_{1 \leq j \leq m} \left\{ t_1^{(j)} + t_2^{(j)} + \cdots + t_n^{(j)} \right\} \tag{2.4}$$

式 (2.3) と (2.4) より，オンラインアルゴリズム A の競合比を m で抑えることができる.

同様にして，以下が成り立つ.

$$A(I) = \sum_{i=1}^{n} \sum_{j=1}^{m} t_i^{(j)} \leq n \cdot \max_{1 \leq i \leq n} \left\{ t_i^{(1)} + t_i^{(2)} + \cdots + t_i^{(m)} \right\} \tag{2.5}$$

$$OPT(I) \geq \max_{1 \leq i \leq n} \left\{ t_i^{(1)} + t_i^{(2)} + \cdots + t_i^{(m)} \right\} \tag{2.6}$$

式 (2.5) と (2.6) より，オンラインアルゴリズム A の競合比を n で抑えることができる.

以上から，オンラインアルゴリズム A の競合比は $\min\{m, n\}$ である. (証明終)

例題 5. オンラインアルゴリズム A の競合比に関して，タイトな例を示す．入力 I が $t_i^{(j)} = 1$ $(1 \leq i \leq n, 1 \leq j \leq m)$ のとき,

$$A(I) = mn$$

$$OPT(I) = m + n - 1$$

である．よって，競合比に関して，以下の関係式が導かれる．$m < n$ のとき,

$$\frac{mn}{m+n-1} = \frac{m}{m/n + 1 - 1/n} \xrightarrow[(n \to \infty)]{} m.$$

$m > n$ のとき,

$$\frac{mn}{m+n-1} = \frac{n}{1 + n/m - 1/m} \xrightarrow[(m \to \infty)]{} n.$$

また，$m = n$ のときは,

$$\frac{mn}{m+n-1} = \frac{m^2}{2m-1} \leq m.$$

2.3 製造装置の表記法

ここでは，実在する製造装置を記号化する．スケジューリング理論では，Graham らが最初に提案した表記法 [51] を用いて，スケジューリング問題を記述する．Graham らの表記法によると，基本的には資源（機械）の特性，ジョブの特性，目的関数の 3 つの要素でスケジューリング問題を分類する．[4] では，Graham らが提案したものを，最近の研究成果も踏まえて拡張した表記法を紹介している．

また，待ち行列理論では，ケンドールの記号 [17] を用いて，待ち行列モデルの分類を行う．この記法は分かりやすく便利であり，単純な待ち行列モデルを分類するのに優れている．ケンドールの記法では，主に搬送系への到着分布型，処理時間の分布型，機械の数で待ち行列モデルを表す．

一方，実存する製造装置は，独自の制約を持つ．そのため，ケンドールの記号のみで実在の製造装置を分類することができない．我々は，それらの製造装置の独自の制約を以下に示す要素で分類する．

1. 到着分布
   (a) タクト型 (T: tact): 搬送系への到着分布が単位分布．ほとんどの製造装置では，タクト型を使っており，入口での搬送制御も容易．
   (b) 非同期型 (A: asynchronous): 搬送系への到着到着分布が単位分布ではない．ジョブの投入時刻の間隔が異なることを許すため，タクト型に比べると入口での搬送制御が複雑．
2. 機械間の接続関係
   (a) 直列型 (S: serial): 機械が一列に並ぶ．
   (b) 並列型 (P: parallel): 複数の同一機械が並列に並ぶ．図 2.10 に並列型の例を示す．
   (c) ネットワーク型 (N: network): 一般の接続型．機械を点，接続関係を枝と見なすとネットワークとして見ることができる．図 2.11 にネットワーク型の例を示す．
3. キャリアの搬送時間

(a) 0 型: 搬送時間がない (すなわち搬送時間が 0 である).

(b) C 型 (C: constant): 搬送時間が定数 (> 0). ただし，キャリア毎に搬送時間が異なってもよい.

(c) A 型 (A: acceleration): 移動時の加速度を考慮する.

4. ジョブの物理サイズ

(a) P 型 (P: point): ジョブは点である.

(b) L 型 (L: length): ジョブは長さのみを持つ.

(c) R 型 (R: rectangle): ジョブは長方形である.

5. 終了センサー

処理の完了を探知する装置を終了センサーと呼ぶ. 終了センサーで探知した情報を前の機械に伝えることで，ジョブ同士の衝突回避に利用する.

(a) N 型 (N: no): 終了センサーが無い.

(b) D 型 (D: detectable): 探知型. 各機械に終了センサーが準備されている.

(c) A 型 (A: advanced): あとどれくらいで処理が完了するかを探知できる終了センサー (advance detection) を，各機械に付けている.

上記の 1 から 5 までを下のように並べて表現する. ただし，最初の m は機械数であり，その後は順番に上記 1 から 5 に対応する.

$$m \left(\begin{array}{c} T \\ A \end{array} \right) \left(\begin{array}{c} S \\ P \\ N \end{array} \right) \left(\begin{array}{c} 0 \\ C \\ A \end{array} \right) \left(\begin{array}{c} P \\ L \\ R \end{array} \right) \left(\begin{array}{c} N \\ D \\ A \end{array} \right)$$

(例) $m(T)(S)(0)(P)(N)$: m 機械，タクト型，直列型，搬送時間が 0，ジョブを点とみなす，終了センサー無しの製造装置待ち行列モデル.

$6(T)(P)(0)(P)(N)$: 6 機械，タクト型，並列型，搬送時間が 0，ジョブを点とみなす，終了センサー無しの製造装置待ち行列モデル.

$9(T)(N)(0)(P)(N)$: 9 機械，タクト型，ネットワーク型，搬送時間が 0，ジョブを点とみなす，終了センサー無しの製造装置待ち行列モデル.

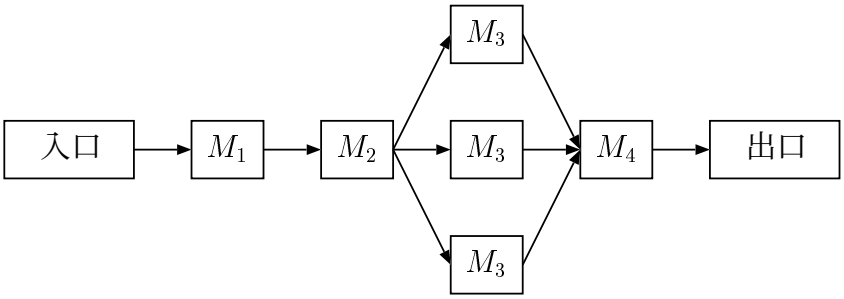


図 2.10: 並列型

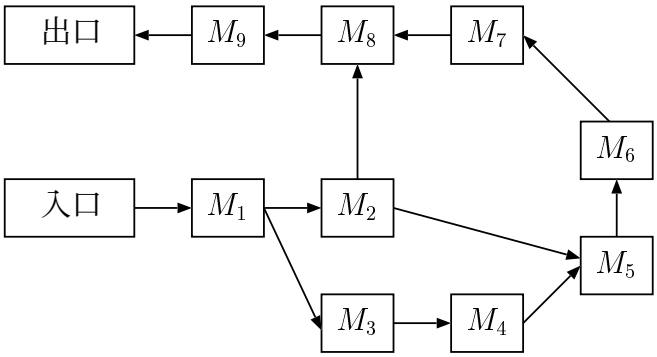


図 2.11: ネットワーク型

第 3章

直列型の搬送系

3.1　1機械モデル

本章では，処理時間を確率的に扱う．単純な例として，図 3.1 に示すような 1 台の機械からなる搬送系から議論する．ジョブは入口から投入されて，機械 M_1 は運ばれてきたジョブを処理する．そして M_1 での処理が終了するとすぐに出口へと運ばれる．入口からのジョブの投入は一定の周期（T_{tact}）で行われる．処理時間に関して，本章では正規分布（ガウス分布）に従う場合と，指数分布に従う場合を分けて説明する．実際は処理時間は正規分布に従うことが経験的に知られているが，解析結果はあまりすっきりしない形になる．一方，指数分布に従うと仮定すると（正規分布のときと比べて）理論的な解析結果が明示的に示される．

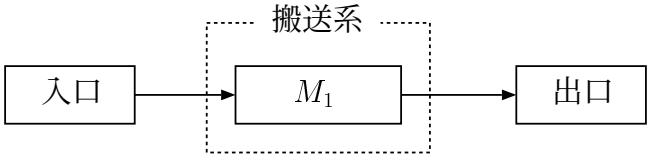


図 3.1: 1 台の機械からなる搬送系.

3.1.1 処理時間が正規分布に従う場合

ここでは，機械 M_1 でのジョブ J_i $(1 \leq i \leq n)$ の処理時間 t_i [1] を確率変数として扱う．すなわち，確率変数 t_i $(1 \leq i \leq n)$ は，平均値が μ で標準偏差が σ の正規分布に従う（しばしば，$t_i \sim \mathrm{N}(\mu, \sigma^2)$ と記述される）と仮定する．一般に，連続型確率変数 X が正規分布 $\mathrm{N}(\mu, \sigma^2)$ に従うとき，X の確率密度関数 $f(x)$ は

$$f(x) = \frac{1}{\sqrt{2\,\pi}\,\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\,\sigma^2}\right\}$$

で与えられる [6]．それゆえ，特に X が正規分布 $\mathrm{N}(0,1)$ に従うとき

$$f(x) = \frac{1}{\sqrt{2\,\pi}} \exp\left(-\frac{x^2}{2}\right)$$

である．また，t_i $(1 \leq i \leq n)$ は互いに独立であると仮定する．

衝突確率を解析する．補題 1 [2] より，n 個のジョブが衝突しない確率は

$$\begin{aligned}
&\Pr\left(t_1 \leq T_{\mathrm{tact}}\,,\, t_2 \leq T_{\mathrm{tact}}\,,\, \ldots\,,\, t_{n-1} \leq T_{\mathrm{tact}}\right) \\
&= \left\{\Pr(t_1 \leq T_{\mathrm{tact}})\right\}^{n-1} \qquad (3.1) \\
&\qquad (\because\ t_i\,(1 \leq i \leq n-1) \text{ は互いに独立で同一の分布に従う.}) \\
&= \left\{\Pr\left(t \leq \frac{T_{\mathrm{tact}} - \mu}{\sigma}\right)\right\}^{n-1} \qquad (3.2) \\
&\qquad \text{ここで } t \sim \mathrm{N}(0,1) \qquad \left(\because \text{標準化変数 } t = \frac{t_1 - \mu}{\sigma}\right) \\
&= \left\{\int_{-\infty}^{\frac{T_{\mathrm{tact}}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx\right\}^{n-1}.
\end{aligned}$$

で与えられる．式 (3.1) は，確率変数 t_i が一般の分布に従うときでも成り立つ．式 (3.2) は確率変数 t_i が特に正規分布に従うことを利用して，標準化したものである．

“n 個のジョブが衝突しない” という事象の余事象は，“衝突が存在する” という事象である．それゆえ，$n \geq 1$ に対して衝突確率は以下のようになる．

$$1 - \left\{\int_{-\infty}^{\frac{T_{\mathrm{tact}}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx\right\}^{n-1}. \qquad (3.3)$$

[1] M_1 での J_i の処理時間を表しているので，第 2 章で示した表記に従うと，t_i ではなく $t_i^{(1)}$ と書くべきである．しかし，本章では 1 機械の場合しか扱わないので，t_i と略記する．

[2] 補題 1 は，第 2 章で述べた．

また，チェビシェフの不等式を用いても同じような解析が可能であるが，数値的には具体的な分布を仮定して積分値を求める上の方式の方が信頼できる．

式 (3.3) 中の積分を初等関数[3] を使って表すことはできない[4] [11] が，下のようにガウスの誤差関数を使って表すことはできる．

$$\text{式}\,(3.3) = 1 - \left\{ \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{T_{\text{tact}} - \mu}{\sqrt{2}\,\sigma} \right) \right) \right\}^{n-1}.$$

ただし，上式でガウスの誤差関数 $\operatorname{erf}(z)$ は

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$$

で与えられる[5] [52]．誤差関数の近似値は，ヘイスティングスの近似値などを利用すれば，速くかつ高い精度で計算することができる [3]．

タクトタイム T_{tact} を特に $\mu+\sigma$, $\mu+2\sigma$ 及び $\mu+3\sigma$ にそれぞれ設定したとき，衝突確率（すなわち，式 (3.3) ）は次のようになる．式 (3.2) から，衝突が起きない確率は，それぞれ $(\Pr(t \le 1))^{n-1} \approx 0.84134^{n-1}, (\Pr(t \le 2))^{n-1} \approx 0.97725^{n-1}$ 及び $(\Pr(t \le 3))^{n-1} \approx 0.99865^{n-1}$ となる．それゆえ，衝突確率は以下のようになる．

$$\begin{aligned}
&1 - 0.84134^{n-1} && (T_{\text{tact}} = \mu + \sigma \text{ のとき}),\\
&1 - 0.97725^{n-1} && (T_{\text{tact}} = \mu + 2\sigma \text{ のとき}),\\
&1 - 0.99865^{n-1} && (T_{\text{tact}} = \mu + 3\sigma \text{ のとき}).
\end{aligned}$$

次に，式 (3.3) の解析結果を使って，下に述べるような実用的な問題を考える．

実用的な問題. 衝突確率が与えられた定数 α $(0 \le \alpha \le 1)$ 以下になるという条件下で，タクトタイム T_{tact} を最小化せよ．

上の問題は，式 (3.2) を利用すれば容易に求めることができる．すなわち，次の不等式をタクトタイム T_{tact} に関して解けばよい．

$$1 - \left\{ \Pr\left(t \le \frac{T_{\text{tact}} - \mu}{\sigma} \right) \right\}^{n-1} \le \alpha.$$

[3][8] によると，代数関数，指数関数，対数関数，3 角関数，逆 3 角関数，およびこれらの関数の有限回の合成で得られる，実数または複素数変数の関数を初等関数という．

[4] リュービルが e^{-x^2} の不定積分は初等関数で表されないことを証明したから（例えば，[2] などを参照）．参考までに述べるが，[8] によると，初等関数の積分がどういうとき初等関数となるかについてはリュービルの研究がある [29] [5]，とのこと．

[5] 誤差関数の定義は書物によって異なるので注意する．例えば，[11] や [12] では，$\operatorname{erf}(z) = \int_0^z e^{-t^2}\, dt$ のように，係数 $\frac{2}{\sqrt{\pi}}$ を除いた形で定義されている．

上の不等式中で n, μ, σ, α は定数である．この不等式を変形すれば，次のようになる．

$$\Pr\left(t \leq \frac{T_{\text{tact}} - \mu}{\sigma}\right) \geq (1-\alpha)^{\frac{1}{n-1}}.$$

ここで，正規分布の累積分布関数（CDF）が $(1-\alpha)^{\frac{1}{n-1}}$ に達する点の値を β とすれば，次の不等式を満たす T_{tact} を求めればよい．

$$\frac{T_{\text{tact}} - \mu}{\sigma} \geq \beta$$

この不等式を変形すれば，次のようになる．

$$T_{\text{tact}} \geq \mu + \beta\sigma.$$

それゆえ，$T_{\text{tact}} = \mu + \beta\sigma$ とすればよい．

例題 6. ジョブの個数が 40 で，衝突確率が 0.05 以下になるように（すなわち，$n = 40$，$\alpha = 0.05$）タクトタイムを求める．以下の関係がある．

$$(1-\alpha)^{\frac{1}{n-1}} = 0.95^{\frac{1}{39}} \approx 0.998686.$$

また，正規分布の累積分布関数が 0.998686 に達する点の値は，ほぼ 3.0082 [6] となる．それゆえ，

$$T_{tact} = \mu + 3.0082\,\sigma$$

となる．

3.1.2 処理時間が指数分布に従う場合

処理時間が指数分布に従う場合を考える．すなわち，ジョブ J_i $(1 \leq i \leq n)$ の処理時間 t_i は，一定のパラメータ λ の指数分布に従い，t_i $(1 \leq i \leq n)$ は互いに独立であると仮定する．一般に連続型確率変数 X が指数分布に従うとき，X の確率密度関数 $f(x)$ は

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & (x \geq 0) \ \ (\lambda : \text{正定数}), \\ 0 & (x < 0) \end{cases} \tag{3.4}$$

[6] この値は，Mathematica [37] を使って容易に求めることができる．

で，次の平均 $E(X)$，分散 $V(X)$，標準偏差 σ_X をもつ [6].

$$E(X) = \frac{1}{\lambda}, \qquad V(X) = \frac{1}{\lambda^2}, \qquad \sigma_X = \sqrt{V(X)} = \frac{1}{\lambda}.$$

さて，衝突確率を解析する．n 個のジョブが衝突しない確率は，一般の分布に対して式 (3.1) で与えられる．それゆえ，衝突が存在しない確率は次のようになる．

$$\begin{aligned} \text{式 (3.1)} &= \left(\int_{-\infty}^{T_{\text{tact}}} f(x)\, dx \right)^{n-1} \\ &= \left(\int_{0}^{T_{\text{tact}}} \lambda\, e^{-\lambda\, x}\, dx \right)^{n-1} \\ &= \left(1 - e^{-\lambda T_{\text{tact}}} \right)^{n-1}. \end{aligned}$$

それゆえ，衝突確率は次の式のようになる．

$$1 - \left(1 - e^{-\lambda T_{\text{tact}}} \right)^{n-1}. \tag{3.5}$$

ここで注目すべきことは，衝突確率を初等関数で明示的に示せたことである．一方，処理時間が正規分布に従う場合は，式 (3.3) に示したように明示的に示せない．

さて，再び衝突確率がある定数 α $(0 \leq \alpha \leq 1)$ 以下という制約の下でタクト T_{tact} を最小化する問題（実用的な問題）を考える．上で求めた式 (3.5) を使って，次の不等式をタクトタイム T_{tact} に関して解けばよい．

$$1 - \left(1 - e^{-\lambda T_{\text{tact}}} \right)^{n-1} \leq \alpha.$$

上の不等式中で n, λ, α は定数である．上の不等式を変形すれば，次のようになる．

$$T_{\text{tact}} \geq -\frac{1}{\lambda} \ln \left(1 - (1-\alpha)^{\frac{1}{n-1}} \right).$$

それゆえ，$T_{\text{tact}} = -\frac{1}{\lambda} \ln \left(1 - (1-\alpha)^{\frac{1}{n-1}} \right)$ とすればよい[7]．このようにして，衝突確率がある与えられた値以下になるという制約の下で最適な（最小の）タクトを求めることができる．

3.2 2 機械モデル

第 3.1 章では，1 台の機械からなる搬送系を議論した．ここでは，図 3.2 に示すような 2 台の機械からなる搬送系を議論する．ジョブは入口から投入され，最初の

[7] ln は対数関数を示す．$\ln n = \log_e n$（自然対数）のことである．本論文を通して，標準的な数学関数と記号については，[53] に従う．

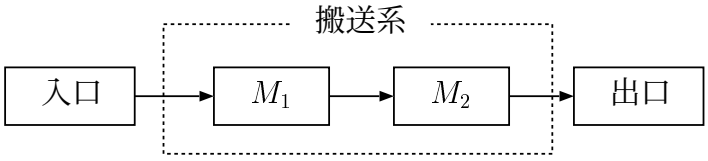


図 3.2: 2 台の機械からなる搬送系.

機械 M_1 へと運ばれる．M_1 へ到着すると，すぐに M_1 はジョブの処理を開始する．処理が完了すると，すぐに次に機械 M_2 へと運ばれる．M_2 へ到着すると，すぐに M_2 はジョブの処理を開始し，処理が完了すると，すぐに出口へと運ばれる．入口からのジョブの投入間隔は一定で，タクトタイム（$T_{\rm tact}$）の間隔で投入される．

処理時間に関して，第 3.1 章と同じように，正規分布に従う場合と指数分布に従う場合に分けて説明する．第 3.1 章で行った解析をそのまま 2 台の機械からなる搬送系に適用しようとすると，途中で破綻するが，変数変換をうまく利用することで衝突確率を導く．

3.2.1 処理時間が正規分布に従う場合

ここでは，機械 M_1 でのジョブ J_i $(1 \leq i \leq n)$ の処理時間 $t_i^{(1)}$ が正規分布 $\mathrm{N}(\mu_1, \sigma_1^2)$ に従い，機械 M_2 でのジョブ J_i $(1 \leq i \leq n)$ の処理時間 $t_i^{(2)}$ が正規分布 $\mathrm{N}(\mu_2, \sigma_2^2)$ に従うと仮定する．また，$t_i^{(j)}$ $(i = 1, 2, \ldots, n,\ j = 1, 2)$ は互いに独立であると仮定する．

衝突確率を解析する．そのために，2 つの確率変数 X_1 と X_2 を

$$X_1 = t_i^{(1)} - T_{\rm tact}\ ,$$
$$X_2 = t_i^{(1)} + t_i^{(2)} - t_{i+1}^{(1)} - T_{\rm tact}$$

とおく．補題 2 [8] より，ジョブ J_i と J_{i+1} $(1 \leq i \leq n-1)$ が衝突しない確率は，次

[8] 補題 2 は，第 2 章で述べた.

のようになる．

$$
\begin{aligned}
&\Pr\left(t_i^{(1)} \le T_{\rm tact}\,,\, t_i^{(1)} + t_i^{(2)} \le t_{i+1}^{(1)} + T_{\rm tact}\right) \\
&= \Pr\left(X_1 \le 0\,,\, X_2 \le 0\right) \\
&= \iint_D f(x_1, x_2)\, dx_1\, dx_2\,, \qquad (3.6) \\
&D : x_1 \le 0\,,\, x_2 \le 0.
\end{aligned}
$$

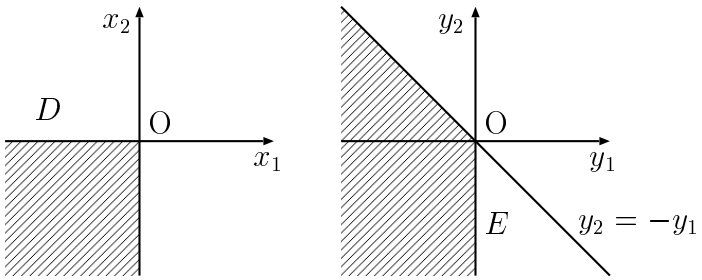
ただし，上式中で $f(x_1, x_2)$ は X_1 と X_2 の同時確率密度関数を示す．確率変数 X_1 と X_2 は互いに独立ではない．そこで，次のような変数変換を行う．

$$
\text{変換} \begin{cases} y_1 = x_1 \\ y_2 = x_2 - x_1 \end{cases} \qquad (x_1 = y_1,\ x_2 = y_1 + y_2)
$$

によって，

$$
E : y_1 \le 0\,,\, y_1 + y_2 \le 0
$$

と，D とは一対一に対応する．



$x_1 = y_1, x_2 = y_1 + y_2$ のヤコビアン $\mathscr{J}$ は

$$
\mathscr{J} = \frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix} = 1
$$

であるから，次の式が得られる．

$$
\begin{aligned}
\text{式 } (3.6) &= \iint_E g(y_1, y_2)\, |\mathscr{J}|\, dy_1\, dy_2 \\
&= \int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} g(y_1, y_2)\, dy_2\,. \qquad (3.7)
\end{aligned}
$$

ただし，上式中の $g(y_1, y_2)$ は確率変数 Y_1 と Y_2 の同時確率密度関数を示し，

$$Y_1 = X_1 = t_i^{(1)} - T_{\text{tact}}\,,$$
$$Y_2 = X_2 - X_1 = t_i^{(2)} - t_{i+1}^{(1)}$$

である．それゆえ，Y_1 と Y_2 は互いに独立である．式 (3.7) は処理時間が一般の分布に従うときでさえ成り立つ．

さて，処理時間が正規分布に従うことを利用すると，Y_1 は $\mathrm{N}(\mu_1 - T_{\text{tact}}, \sigma_1^2)$ に従い，Y_2 は正規分布の再生性[9] より，$\mathrm{N}(\mu_2 - \mu_1, \sigma_1^2 + \sigma_2^2)$ に従う．Y_1 と Y_2 の確率密度関数をそれぞれ $g_1(y_1), g_2(y_2)$ とすると，Y_1, Y_2 は独立なので，式 (3.7) 中の $g(y_1, y_2)$ は

$$\begin{aligned}
&g(y_1, y_2)\\
&= g_1(y_1) \cdot g_2(y_2)\\
&= \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left\{-\frac{(y_1 - \mu_1 + T_{\text{tact}})^2}{2\sigma_1^2}\right\} \cdot \frac{1}{\sqrt{2\pi}\sqrt{\sigma_1^2 + \sigma_2^2}} \exp\left\{-\frac{(y_2 - \mu_2 + \mu_1)^2}{2(\sigma_1^2 + \sigma_2^2)}\right\}\\
&= \frac{1}{2\pi\sigma_1\sqrt{\sigma_1^2 + \sigma_2^2}} \exp\left\{-\frac{1}{2}\left(\frac{(y_1 - \mu_1 + T_{\text{tact}})^2}{\sigma_1^2} + \frac{(y_2 - \mu_2 + \mu_1)^2}{\sigma_1^2 + \sigma_2^2}\right)\right\}
\end{aligned}$$

となる．それゆえ，式 (3.7) を明示的に示す（初等関数で表す）ことができないだろう．しかし，式 (3.7) の数値的な値（近似値）を計算することができる[10] ．補題 2 より，n 個のジョブが衝突を起こさない確率は

$$\left\{\Pr\left(t_i^{(1)} \le T_{\text{tact}}\,,\, t_i^{(1)} + t_i^{(2)} \le t_{i+1}^{(1)} + T_{\text{tact}}\right)\right\}^{n-1}$$

（すなわち，$(\text{式 (3.7)})^{n-1}$）であるから，求めるべき衝突確率は次のようになる．

$$1 - \left(\int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} g(y_1, y_2)\, dy_2\right)^{n-1}. \tag{3.8}$$

次に，式（3.8）の解析結果を使って，下に示した実用的な問題を考える．

実用的な問題. 衝突確率が与えられた定数 α $(0 \le \alpha \le 1)$ 以下になるという条件下で，タクトタイム T_{tact} を最小化せよ．

[9] 正規分布の再生性: 一般に 2 つの確率変数 X，Y が独立で，それぞれ $\mathrm{N}(\mu_1, \sigma_1^2)$，$\mathrm{N}(\mu_2, \sigma_2^2)$ に従っているとき，$X + Y$ は $\mathrm{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$ に従う．例えば，[6] などを参照されたい．

[10] MATHEMATICA を利用しても近似値を求めることができる．

衝突確率は式 (3.8) で与えられる．それゆえ，次の不等式

$$1 - \left(\int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} g(y_1, y_2)\, dy_2 \right)^{n-1} \leq \alpha$$

をタクトタイムに関して解けばよいが，これを直接（式変形で）求めるのは難しい．実用的な問題を考えるときに注意するのは，式 (3.8) 中でタクトタイム $T_{\rm tact}$ のみが変数であり，そのほかの n, m, μ_1, μ_2, σ_1, σ_2 は定数だということである．衝突確率（すなわち，式 (3.8) ）は，タクトタイムが短いほど高くなり，逆にタクトタイムを長くすれば長くするほど小さくなる．実際，式 (3.8) はタクトタイムに関して強い意味で単調減少である[11] ．そのため，タクトタイムに関して二分探索法を使って，最適な（最小の）タクトタイムを求めることができる．二分探索法の過程で式 (3.8) 中の重積分の値を求めなくてはならない．

3.2.2 処理時間が指数分布に従う場合

処理時間が指数分布に従う場合の衝突確率を解析する．ジョブ J_i $(1 \leq i \leq n)$ の機械 M_1 での処理時間 $t_i^{(1)}$ は，一定のパラメータ λ_1 の指数分布に従い，ジョブ J_i $(1 \leq i \leq n)$ の機械 M_2 での処理時間 $t_i^{(2)}$ は，一定のパラメータ λ_2 の指数分布に従う．それゆえ，M_1 と M_2 での処理時間はそれぞれ λ_1 と λ_2 に依存している．

ジョブ J_i と J_{i+1} $(1 \leq i \leq n-1)$ が衝突しない確率は式 (3.7) で与えられる．なぜなら，式 (3.7) は一般の分布に対して成り立つからである．ここで，処理時間が指数分布に従うことを利用する．読者のために，もう一度確率変数 Y_1 と Y_2 を下に示す．

$$Y_1 = t_i^{(1)} - T_{\rm tact}\ ,$$
$$Y_2 = t_i^{(2)} - t_{i+1}^{(1)}.$$

それゆえ，Y_1 の確率密度関数 $g_1(y_1)$ は

$$g_1(y_1) = \begin{cases} \lambda_1\, e^{-\lambda_1 (y_1 + T_{\rm tact})} & (y_1 \geq -T_{\rm tact}), \\ 0 & (y_1 < -T_{\rm tact}). \end{cases}$$

[11] 一般に，関数 $f(n)$ が強い意味で単調減少であるのは，$m < n$ なら $f(m) > f(n)$ が成り立つときである．

である．一方，Y_2 の確率密度関数 $g_2(y_2)$ は以下のようにして導かれる．確率変数 $t_i^{(2)}$ の確率密度関数を $h_1(x)$ とすると，

$$h_1(x) = \begin{cases} \lambda_2\, e^{-\lambda_2 x} & (x \geq 0), \\ 0 & (x < 0) \end{cases}$$

であり，$-t_{i+1}^{(1)}$ の確率密度関数を $h_2(y)$ とすると，

$$h_2(y) = \begin{cases} 0 & (y > 0), \\ \lambda_1\, e^{\lambda_1 y} & (y \leq 0) \end{cases}$$

である．ゆえに，畳込みの定理[12] より，Y_2 の確率密度関数 $g_2(y_2)$ は以下のようになる．

Case 1: $y_2 \geq 0$ のとき

$$\begin{aligned} g_2(y_2) &= \int_{-\infty}^{\infty} h_1(x)\, h_2(y_2 - x)\, dx \\ &= \int_{y_2}^{\infty} \lambda_2 e^{-\lambda_2 x} \cdot \lambda_1 e^{\lambda_1 (y_2 - x)} dx \\ &= \int_{y_2}^{\infty} \lambda_1 \lambda_2 e^{\lambda_1 y_2} e^{-(\lambda_1 + \lambda_2) x} dx \\ &= \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} e^{-\lambda_2 y_2}\,, \end{aligned} \tag{3.9}$$

Case 2: $y_2 < 0$ のとき

$$\begin{aligned} g_2(y_2) &= \int_{-\infty}^{\infty} h_1(x)\, h_2(y_2 - x)\, dx \\ &= \int_{0}^{\infty} \lambda_1 \lambda_2 e^{\lambda_1 y_2} e^{-(\lambda_1 + \lambda_2) x} dx \\ &= \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} e^{\lambda_1 y_2}. \end{aligned} \tag{3.10}$$

[12] 畳込みの定理：確率変数 X，Y が独立で，それらの確率密度関数を $f(x)$，$g(y)$ とする．このとき，和 $Z = X + Y$ の確率密度関数 $h(z)$ は

$$h(z) = \int_{-\infty}^{\infty} f(x)\, g(z - x)\, dx$$

である．例えば，[6] などを参照されたい．

式 (3.9) と (3.10) をまとめると，

$$g_2(y_2) = \begin{cases} \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2} e^{-\lambda_2 y_2} & (y_2 \geq 0), \\ \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2} e^{\lambda_1 y_2} & (y_2 < 0) \end{cases}$$

となる．

以上で，J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.7) ）を計算する準備が整った．

$$\begin{aligned} \text{式}\,(3.7) &= \int_{-\infty}^{0} \left(\int_{-\infty}^{-y_1} g_1(y_1) g_2(y_2) dy_2 \right) dy_1 \quad (\because\ Y_1, Y_2 \text{ は独立}) \\ &= \int_{-\infty}^{0} \left(g_1(y_1) \int_{-\infty}^{-y_1} g_2(y_2) dy_2 \right) dy_1 \,. \end{aligned} \tag{3.11}$$

ここで，

$$\begin{aligned} \int_{-\infty}^{-y_1} g_2(y_2) dy_2 &= \int_{-\infty}^{0} \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2} e^{\lambda_1 y_2} dy_2 + \int_{0}^{-y_1} \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2} e^{-\lambda_2 y_2} \\ &= 1 - \frac{\lambda_1}{\lambda_1+\lambda_2} e^{\lambda_2 y_1} \end{aligned}$$

であるから，

$$\begin{aligned} \text{式}\,(3.11) &= \int_{-\infty}^{0} g_1(y_1) \left(1 - \frac{\lambda_1}{\lambda_1+\lambda_2} e^{\lambda_2 y_1} \right) dy_1 \\ &= \int_{-\infty}^{-T_{\text{tact}}} g_1(y_1) \left(1 - \frac{\lambda_1}{\lambda_1+\lambda_2} e^{\lambda_2 y_1} \right) dy_1 \\ &\quad + \int_{-T_{\text{tact}}}^{0} g_1(y_1) \left(1 - \frac{\lambda_1}{\lambda_1+\lambda_2} e^{\lambda_2 y_1} \right) dy_1 \\ &= \int_{-T_{\text{tact}}}^{0} \lambda_1 e^{-\lambda_1 (y_1 + T_{\text{tact}})} \left(1 - \frac{\lambda_1}{\lambda_1+\lambda_2} e^{\lambda_2 y_1} \right) dy_1 \\ &= \int_{-T_{\text{tact}}}^{0} \left(\lambda_1 e^{-\lambda_1 (y_1 + T_{\text{tact}})} - \frac{\lambda_1^2}{\lambda_1+\lambda_2} e^{-\lambda_1 T_{\text{tact}}} \cdot e^{-(\lambda_1-\lambda_2) y_1} \right) dy_1 \\ &= \begin{cases} 1 + \frac{e^{-\lambda_1 T_{\text{tact}}} \lambda_2^2}{\lambda_1^2-\lambda_2^2} + \frac{e^{-\lambda_2 T_{\text{tact}}} \lambda_1^2}{\lambda_2^2-\lambda_1^2} & (\lambda_1 \neq \lambda_2 \text{のとき}) \,, \\ 1 - e^{-\lambda T_{\text{tact}}} - \frac{\lambda}{2} e^{-\lambda T_{\text{tact}}} \cdot T_{\text{tact}} & (\lambda = \lambda_1 = \lambda_2 \text{のとき}) \,. \end{cases} \end{aligned}$$

上式のように，ジョブ J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.7) ）を明示的に（初等関数を使って）示すことができた．それゆえ，ジョブの個数が n の

とき，求めるべき衝突確率は

$$1-\left(1+\frac{e^{-\lambda_1 T_{\rm tact}}\lambda_2^2}{\lambda_1^2-\lambda_2^2}+\frac{e^{-\lambda_2 T_{\rm tact}}\lambda_1^2}{\lambda_2^2-\lambda_1^2}\right)^{n-1} \qquad \lambda_1 \neq \lambda_2 \text{ のとき}, \tag{3.12}$$

$$1-\left(1-e^{-\lambda T_{\rm tact}}-\frac{\lambda}{2}e^{-\lambda T_{\rm tact}}\cdot T_{\rm tact}\right)^{n-1} \qquad \lambda=\lambda_1=\lambda_2 \text{ のとき} \tag{3.13}$$

となる．

さて，再び衝突確率がある定数α $(0 \leq \alpha \leq 1)$以下という制約の下でタクト$T_{\rm tact}$を最小化する問題（実用的な問題）を考える．衝突確率は式 (3.12) もしくは (3.13) で与えられる．それゆえ，次の不等式

$$1-\left(1+\frac{e^{-\lambda_1 T_{\rm tact}}\lambda_2^2}{\lambda_1^2-\lambda_2^2}+\frac{e^{-\lambda_2 T_{\rm tact}}\lambda_1^2}{\lambda_2^2-\lambda_1^2}\right)^{n-1} \leq \alpha$$

もしくは

$$1-\left(1-e^{-\lambda T_{\rm tact}}-\frac{\lambda}{2}e^{-\lambda T_{\rm tact}}\cdot T_{\rm tact}\right)^{n-1} \leq \alpha$$

をタクトタイムに関して解けばよい．実用的な問題を考えるときに注意するのは，式 (3.12)，(3.13) 中でタクトタイム$T_{\rm tact}$のみが変数であり，そのほかのn, λ_1, λ_2は定数である．$m=1$のときは第3.1章で述べたように，上の不等式をタクトタイムに関して式変形で解くことができる．今回の場合は，式変形で解くのは難しい．以前にも述べたように，衝突確率はタクトタイムが短いほど高くなり，逆にタクトタイムを長くすれば長くするほど小さくなる．実際，式 (3.12) と (3.13) はタクトタイムに関して強い意味で単調減少である．そのため，タクトタイムに関して二分探索法を使って，最適な（最小の）タクトタイムを求めることができる．

3.3 m機械モデル

ここでは，図1.1 [13] に示すような一般のm台の機械からなる搬送系を議論する．第3.2章での解析をそのまま拡張することで，m台の場合でも容易に衝突確率を解析することができる．ジョブは入口から投入され，最初の機械M_1へと運ばれる．ジョブが機械M_j $(1 \leq j \leq m-1)$へ到着すると，すぐにM_jは到着したジョブの処理を開始する．その処理が終了すると，すぐに次の機械M_{j+1}へと運ばれる．最

[13] この図は第1章で示した．

後の機械 M_m に到着すると，すぐに M_m は処理を開始し，処理が終了すると，この搬送系から退出する．搬送系へのジョブの投入間隔を一定とする（すなわち，タクトタイムの間隔で投入される）．

処理時間に関してはこれまでと同様に，正規分布に従う場合と指数分布に従う場合に分けて説明する．また，たくさんの例題を示すことで，実際に計算する際に役立つものと考える．最後に，実用的な問題の解法を考察する．

3.3.1 処理時間が正規分布に従う場合

機械 M_j $(1 \leq j \leq m)$ でのジョブ J_i $(1 \leq i \leq n)$ の処理時間 $t_i^{(j)}$ は，正規分布 $\mathrm{N}(\mu_j, \sigma_j^2)$ に従うと仮定する．それゆえ，処理時間に関して機械毎に異なる平均値と標準偏差になる場合も考慮している．また，$t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ は互いに独立であると仮定する．

衝突確率を解析する．m 個の確率変数 X_j $(1 \leq j \leq m)$ を

$$X_j = \sum_{k=1}^{j} t_i^{(k)} - \sum_{k=1}^{j-1} t_{i+1}^{(k)} - T_{\mathrm{tact}}$$

とおく．補題 3 [14] より，ジョブ J_i と J_{i+1} $(1 \leq i \leq n-1)$ が衝突しない確率は，次のようになる．

$$\begin{aligned}
&\Pr\left(\text{すべての } j \ (1 \leq j \leq m) \text{ に対して} \quad \sum_{k=1}^{j} t_i^{(k)} \leq T_{\mathrm{tact}} + \sum_{k=1}^{j-1} t_{i+1}^{(k)}\right) \\
&= \Pr\left(X_1 \leq 0, X_2 \leq 0, \ldots, X_m \leq 0\right) \\
&= \iint \cdots \int_{S_1} f(x_1, x_2, \ldots, x_m)\, dx_1 dx_2 \cdots dx_m, \qquad (3.14) \\
&S_1 : x_1 \leq 0, x_2 \leq 0, \ldots, x_m \leq 0.
\end{aligned}$$

ただし，上式中で $f(x_1, x_2, \ldots, x_m)$ は X_1, X_2, …, X_m の同時確率密度関数を示す．また X_1, X_2, …, X_m は独立ではない．そこで，次のような変数変換を行う．

$$\text{変換} \begin{cases} y_1 = x_1 \\ \text{すべての } j \ (2 \leq j \leq m) \text{ に対して} \quad y_j = x_j - x_{j-1} \end{cases}$$

[14] 補題 3 は，第 2 章で述べた．

によって，

$$S_1' : \text{すべての } j\ (1 \le j \le m) \text{ に対して} \quad \sum_{i=1}^{j} y_i \le 0$$

と，S_1 とは一対一に対応する．$x_i = \sum_{j=1}^{i} y_j\ \ (i = 1, 2, \ldots, m)$ のヤコビアン $\mathscr{J}$ は

$$\mathscr{J} = \frac{\partial(x_1, x_2, \ldots, x_m)}{\partial(y_1, y_2, \ldots, y_m)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} & \frac{\partial x_1}{\partial y_3} & \cdots & \frac{\partial x_1}{\partial y_m} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} & \frac{\partial x_2}{\partial y_3} & \cdots & \frac{\partial x_2}{\partial y_m} \\ \frac{\partial x_3}{\partial y_1} & \frac{\partial x_3}{\partial y_2} & \frac{\partial x_3}{\partial y_3} & \cdots & \frac{\partial x_3}{\partial y_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial y_1} & \frac{\partial x_m}{\partial y_2} & \frac{\partial x_m}{\partial y_3} & \cdots & \frac{\partial x_m}{\partial y_m} \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{vmatrix} = 1$$

であるから，次の式が得られる．

$$\begin{aligned} \text{式 (3.14)} &= \iint \cdots \int_{S_1'} g(y_1, y_2, \ldots, y_m)\, |\mathscr{J}|\, dy_1 dy_2 \cdots dy_m \\ &= \int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} dy_2 \cdots \int_{-\infty}^{-\sum_{i=1}^{m-1} y_i} g(y_1, y_2, \ldots, y_m) dy_m \,. \end{aligned} \tag{3.15}$$

ただし，上式中の $g(y_1, y_2, \ldots, y_m)$ は確率変数 Y_1, Y_2, …, Y_m の同時確率密度関数を示しており，

$$\begin{aligned} &Y_1 = X_1 = t_i^{(1)} - T_{\text{tact}}\ , \\ &Y_2 = X_j - X_{j-1} = t_i^{(j)} - t_{i+1}^{(j-1)} \qquad (2 \le j \le m) \end{aligned}$$

である．それゆえ，Y_1, Y_2, …, Y_m は独立である．式 (3.15) は処理時間が一般の分布に従うときでさえ成り立つ．

さて，処理時間が正規分布に従うことを利用すると，Y_1 は $\mathrm{N}(\mu_1 - T_{\text{tact}}, \sigma_1^2)$ に従い，Y_j $(2 \le j \le m)$ は正規分布の再生性より，$\mathrm{N}(\mu_j - \mu_{j-1}, \sigma_{j-1}^2 + \sigma_j^2)$ に従う．Y_1, Y_2, …, Y_m の確率密度関数をそれぞれ $g_1(y_1)$, $g_2(y_2)$, …, $g_m(y_m)$ とすると，Y_1, Y_2, …, Y_m は独立なので，式 (3.15) 中の $g(y_1, y_2, \ldots, y_m)$ は

$$\begin{aligned} g(y_1, y_2, \ldots, y_m) &= g_1(y_1) \cdot g_2(y_2) \cdots g_m(y_m) \\ &= \frac{\exp\left\{ -\frac{1}{2} \left(\frac{(y_1 - \mu_1 + T_{\text{tact}})^2}{\sigma_1^2} + \sum_{j=2}^{m} \frac{(y_j - \mu_j + \mu_{j-1})^2}{\sigma_{j-1}^2 + \sigma_j^2} \right) \right\}}{\sigma_1 (\sqrt{2\pi})^m \prod_{j=2}^{m} \sqrt{\sigma_{j-1}^2 + \sigma_j^2}} \end{aligned} \tag{3.16}$$

となる[15]．それゆえ，式 (3.15) を明示的に示す（初等関数で表す）ことはできないと予想される．しかし，式 (3.15) の数値的な値（近似値）を計算することはできる. 補題 3 より，n 個のジョブが衝突を起こさない確率は

$$\left\{ \Pr\left(\text{すべての } j \ (1 \le j \le m) \text{ に対して} \quad \sum_{k=1}^{j} t_i^{(k)} \le T_{\text{tact}} + \sum_{k=1}^{j-1} t_{i+1}^{(k)} \right) \right\}^{n-1}$$

(すなわち，(式 (3.15))$^{n-1}$) であるから，求めるべき衝突確率は次のようになる．

$$1 - \left(\int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} dy_2 \cdots \int_{-\infty}^{-\sum_{i=1}^{m-1} y_i} g(y_1, y_2, \ldots, y_m) dy_m \right)^{n-1} . \qquad (3.17)$$

例題 7. $m = 3$ のとき，式 (3.16) は，

$$\begin{aligned} g(y_1, y_2, y_3) &= g_1(y_1) \cdot g_2(y_2) \cdot g_3(y_3) \\ &= \frac{\exp\left\{ -\frac{1}{2} \left(\frac{(y_1 - \mu_1 + T_{tact})^2}{\sigma_1^2} + \frac{(y_2 - \mu_2 + \mu_1)^2}{\sigma_1^2 + \sigma_2^2} + \frac{(y_3 - \mu_3 + \mu_2)^2}{\sigma_2^2 + \sigma_3^2} \right) \right\}}{2\pi\sigma_1\sqrt{2\pi}\sqrt{\sigma_1^2 + \sigma_2^2}\sqrt{\sigma_2^2 + \sigma_3^2}} \end{aligned}$$

となる．

例題 8. 全ての $1 \le j \le m$ に対して，$N(\mu_j, \sigma_j^2) = N(\mu, \sigma^2)$ のとき（全ての機械における処理時間の平均値と標準偏差が同一のとき），

$$\begin{aligned} g_1(y_1) &= \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left\{ -\frac{(y_1 - \mu + T_{tact})^2}{2\sigma^2} \right\} , \\ g_j(y_j) &= \frac{1}{2\sigma\sqrt{\pi}} \exp\left(-\frac{y_j^2}{4\sigma^2} \right) \qquad (j = 2, 3, \ldots, m \text{ のとき}) \end{aligned}$$

となる．それゆえ，式 (3.16) は

$$\text{式 (3.16)} = \frac{1}{(\sigma\sqrt{\pi})^m \, 2^{m-\frac{1}{2}}} \exp\left\{ -\frac{1}{4\sigma^2} \left(2(y_1 - \mu + T_{tact})^2 + \sum_{j=2}^{m} y_j^2 \right) \right\}$$

となる．

次に，式（3.17）の解析結果を使って，下に示す実用的な問題を考える．

[15] 有限の積 $a_1 a_2 \cdots a_n$ は

$$\prod_{k=1}^{n} a_k$$

と書ける．$n = 0$ のとき，この積の値は 1 と定義される．

実用的な問題. 衝突確率が与えられた定数 α $(0 \leq \alpha \leq 1)$ 以下になるという条件下で，タクトタイム T_{tact} を最小化せよ．

衝突確率は式 (3.17) で与えられる．それゆえ，次の不等式

$$1 - \left(\int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} dy_2 \cdots \int_{-\infty}^{-\sum_{i=1}^{m-1} y_i} g(y_1, y_2, \ldots, y_m) dy_m \right)^{n-1} \leq \alpha$$

をタクトタイムに関して解けばよいが，これを直接（式変形で）求めるのは難しい．実用的な問題を考えるときに注意するのは，式 (3.17) 中でタクトタイム T_{tact} のみが変数であり，そのほかの n, m, μ_j $(1 \leq j \leq m)$, σ_j $(1 \leq j \leq m)$ は定数である．衝突確率はタクトタイムが短いほど高くなり，逆にタクトタイムを長くすれば長くするほど小さくなる．実際，式 (3.17) はタクトタイムに関して強い意味で単調減少である．そのため，タクトタイムに関して二分探索法を使って，最適な（最小の）タクトタイムを求めることができる．

m の値が 2 や 3 のように小さな数のときは，このやり方で実用的な問題を高速に解くことができるが，それよりも m の値が大きくなると，次のような問題が生じる．二分探索法の実行最中に式 (3.17) を何回か計算するが，この式は m 重積分を含んでいる．この m 重積分は m が大きくなると，実際に計算するのが困難になる．著者は Mathematica を使っていくつか試したが，計算誤差のせいで全くでたらめな値が返ってきた．このような理由で，単純な二分探索法で実用的な問題を解くことができない．

そこで，m の値が大きい場合（$m > 3$）は，式 (3.17) を計算して求めるのを止めて，別のやり方で衝突確率を求める．すなわち，第 4 章で述べる計算機シミュレーションで求める．このシミュレーションを使えば，m が大きいときでも，衝突確率を求めることができる．それゆえ，二分探索法を実行できる．このように計算機シミュレーションを利用している場合は，我々は擬似二分探索法と呼ぶ．

計算機シミュレーションで求めた衝突確率は，厳密な値（式で求めた値）ではない．ただし，現場においては厳密な値でなくても，それに近い値であれば十分であると考える．それゆえ，擬似二分探索法を使えばよい．

3.3.2 処理時間が指数分布に従う場合

処理時間が指数分布に従う場合の衝突確率を解析する．ジョブ J_i $(1 \leq i \leq n)$ の機械 M_j $(1 \leq j \leq m)$ での処理時間 $t_i^{(j)}$ は，一定のパラメータ λ_j の指数分布に従う．それゆえ，M_j での処理時間は λ_j に依存している（機械毎に異なるパラメータをとることも考慮している）．

ジョブ J_i と J_{i+1} $(1 \leq i \leq n-1)$ が衝突しない確率は式 (3.15) で与えられる．なぜなら，式 (3.15) は一般の分布に対して成り立つからである．ここで，処理時間が指数分布に従うことを利用する．先に述べたように，$Y_1, Y_2, \ldots, Y_m$ は独立であるから，式 (3.15) 中の

$$g(y_1, y_2, \ldots, y_m) = g_1(y_1) \cdot g_2(y_2) \cdots g_m(y_m)$$

である．ただし，$g_j(y_j)$ $(1 \leq j \leq m)$ は確率変数 Y_j の確率密度関数を示す．第 3.2.2 章で行ったように畳込みの定理を利用する[16] ことで，式 (3.15) 中の $g_j(y_j)$ $(1 \leq j \leq m)$ は次のようになる．

$$g_1(y_1) = \begin{cases} \lambda_1 e^{-\lambda_1 (y_1 + T_{\mathrm{tact}})} & (y_1 \geq -T_{\mathrm{tact}}), \\ 0 & (y_1 < -T_{\mathrm{tact}}), \end{cases}$$

$$g_j(y_j) = \begin{cases} \frac{\lambda_{j-1}\,\lambda_j}{\lambda_{j-1}+\lambda_j} e^{-\lambda_j\, y_j} & (y_j \geq 0), \\ \frac{\lambda_{j-1}\,\lambda_j}{\lambda_{j-1}+\lambda_j} e^{\lambda_{j-1}\, y_j} & (y_j < 0). \end{cases} \qquad (2 \leq j \leq m \text{ のとき})$$

それゆえ，以下の注意で示すように J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.15) ）を明示的に（初等関数で）示すことができる．求めるべき衝突確率は $1 -$ (式 (3.15))$^{n-1}$ で与えられるので，式 (3.15) が明示的に示されていることの意味は大きい．

式 (3.15) は，λ_j $(1 \leq j \leq m)$ の値に依存しているが，特に全ての λ_j $(1 \leq j \leq m)$ が異なる場合と，全ての λ_j $(1 \leq j \leq m)$ が同一である場合にどうなるかを以下に示す．

注意 1. $m \geq 1$ に対して，全ての λ_j $(1 \leq j \leq m)$ が異なるとき，J_i と J_{i+1} が衝突

[16] 詳細な手続きは省略するが，第 3.2.2 章で説明したやり方をすればよい．

しない確率（すなわち，式 (3.15) ）は次のようになる．

$$
\begin{aligned}
&1+(-1)^m\left\{\frac{e^{-\lambda_1 T_{tact}}\lambda_2^2\lambda_3^2\cdots\lambda_{m-1}^2\lambda_m^2}{(\lambda_1^2-\lambda_2^2)(\lambda_1^2-\lambda_3^2)\cdots(\lambda_1^2-\lambda_{m-1}^2)(\lambda_1^2-\lambda_m^2)}+\right.\\
&\frac{e^{-\lambda_2 T_{tact}}\lambda_1^2\lambda_3^2\cdots\lambda_{m-1}^2\lambda_m^2}{(\lambda_2^2-\lambda_1^2)(\lambda_2^2-\lambda_3^2)\cdots(\lambda_2^2-\lambda_{m-1}^2)(\lambda_2^2-\lambda_m^2)}+\cdots\\
&\left.+\frac{e^{-\lambda_m T_{tact}}\lambda_1^2\lambda_2^2\cdots\lambda_{m-2}^2\lambda_{m-1}^2}{(\lambda_m^2-\lambda_1^2)(\lambda_m^2-\lambda_2^2)\cdots(\lambda_m^2-\lambda_{m-2}^2)(\lambda_m^2-\lambda_{m-1}^2)}\right\}\\
&=1+(-1)^m\sum_{k=1}^{m}\left(e^{-\lambda_k T_{tact}}\prod_{j\in I_m\setminus\{k\}}\frac{\lambda_j^2}{\lambda_k^2-\lambda_j^2}\right).
\end{aligned}
$$

ただし，上式で I_m は

$$I_m := \{1, 2, \cdots, m\}$$

である．

注意 2. $m \geq 1$ に対して，全ての λ_j $(1 \leq j \leq m)$ が同一で，$\lambda_j = \lambda$ のとき，J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.15) ）は次のようになる[17] ．

$$
\begin{aligned}
&1-e^{-\lambda T_{tact}}-\frac{e^{-\lambda T_{tact}}}{a_m^{(m)}}\left(a_m^{(m-1)}\lambda T_{tact}+a_m^{(m-2)}\lambda^2 T_{tact}^2+\cdots\right.\\
&\quad\left.+a_m^{(2)}\lambda^{m-2}T_{tact}^{m-2}+a_m^{(1)}\lambda^{m-1}T_{tact}^{m-1}\right)\\
&=1-e^{-\lambda T_{tact}}-\frac{e^{-\lambda T_{tact}}}{a_m^{(m)}}\sum_{j=1}^{m-1}a_m^{(m-j)}\lambda^j T_{tact}^j.
\end{aligned}
$$

[17]有限和 $a_1 + a_2 + \cdots + a_n$ は次のように書くことができる．

$$\sum_{k=1}^{n} a_k .$$

$n = 0$ の場合，和の値は 0 と定義される．

ただし，上式で係数 $a_m^{(j)}$ $(j = 1, 2, \ldots, m)$ は

$$\begin{aligned}
a_m^{(1)} &= 1, \\
a_m^{(2)} &= a_{m-1}^{(2)} + m a_{m-1}^{(1)}, \\
a_m^{(3)} &= a_{m-1}^{(3)} + (m+1) a_{m-1}^{(2)}, \\
a_m^{(4)} &= a_{m-1}^{(4)} + (m+2) a_{m-1}^{(3)}, \\
&\vdots \\
a_m^{(m-1)} &= a_{m-1}^{(m-1)} + (2m-3) a_{m-1}^{(m-2)}, \\
a_m^{(m)} &= 2(m-1) a_{m-1}^{(m-1)} \qquad (m \geq 2 \text{ のとき})
\end{aligned}$$

である[18] .

例題 9. 注意 1 で与えられた式をいくつか具体的に計算してみよう．$m = 4$ のとき

$$1 + \frac{e^{-\lambda_1 T_{tact}} \lambda_2^2 \lambda_3^2 \lambda_4^2}{(\lambda_1^2 - \lambda_2^2)(\lambda_1^2 - \lambda_3^2)(\lambda_1^2 - \lambda_4^2)} + \frac{e^{-\lambda_2 T_{tact}} \lambda_1^2 \lambda_3^2 \lambda_4^2}{(\lambda_2^2 - \lambda_1^2)(\lambda_2^2 - \lambda_3^2)(\lambda_2^2 - \lambda_4^2)}$$
$$+ \frac{e^{-\lambda_3 T_{tact}} \lambda_1^2 \lambda_2^2 \lambda_4^2}{(\lambda_3^2 - \lambda_1^2)(\lambda_3^2 - \lambda_2^2)(\lambda_3^2 - \lambda_4^2)} + \frac{e^{-\lambda_4 T_{tact}} \lambda_1^2 \lambda_2^2 \lambda_3^2}{(\lambda_4^2 - \lambda_1^2)(\lambda_4^2 - \lambda_2^2)(\lambda_4^2 - \lambda_3^2)}$$

となり，$m = 5$ のとき

$$1 - \frac{e^{-\lambda_1 T_{tact}} \lambda_2^2 \lambda_3^2 \lambda_4^2 \lambda_5^2}{(\lambda_1^2 - \lambda_2^2)(\lambda_1^2 - \lambda_3^2)(\lambda_1^2 - \lambda_4^2)(\lambda_1^2 - \lambda_5^2)} - \frac{e^{-\lambda_2 T_{tact}} \lambda_1^2 \lambda_3^2 \lambda_4^2 \lambda_5^2}{(\lambda_2^2 - \lambda_1^2)(\lambda_2^2 - \lambda_3^2)(\lambda_2^2 - \lambda_4^2)(\lambda_2^2 - \lambda_5^2)}$$
$$- \frac{e^{-\lambda_3 T_{tact}} \lambda_1^2 \lambda_2^2 \lambda_4^2 \lambda_5^2}{(\lambda_3^2 - \lambda_1^2)(\lambda_3^2 - \lambda_2^2)(\lambda_3^2 - \lambda_4^2)(\lambda_3^2 - \lambda_5^2)} - \frac{e^{-\lambda_4 T_{tact}} \lambda_1^2 \lambda_2^2 \lambda_3^2 \lambda_5^2}{(\lambda_4^2 - \lambda_1^2)(\lambda_4^2 - \lambda_2^2)(\lambda_4^2 - \lambda_3^2)(\lambda_4^2 - \lambda_5^2)}$$
$$- \frac{e^{-\lambda_5 T_{tact}} \lambda_1^2 \lambda_2^2 \lambda_3^2 \lambda_4^2}{(\lambda_5^2 - \lambda_1^2)(\lambda_5^2 - \lambda_2^2)(\lambda_5^2 - \lambda_3^2)(\lambda_5^2 - \lambda_4^2)}$$

になる.

例題 10. $m = 3$ のとき，J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.15) ）は次のようになる.

$\lambda_1 \neq \lambda_2, \lambda_2 \neq \lambda_3, \lambda_3 \neq \lambda_1$ のとき

$$1 - \frac{e^{-\lambda_1 T_{tact}} \lambda_2^2 \lambda_3^2}{(\lambda_1^2 - \lambda_2^2)(\lambda_1^2 - \lambda_3^2)} - \frac{e^{-\lambda_2 T_{tact}} \lambda_1^2 \lambda_3^2}{(\lambda_2^2 - \lambda_1^2)(\lambda_2^2 - \lambda_3^2)} - \frac{e^{-\lambda_3 T_{tact}} \lambda_1^2 \lambda_2^2}{(\lambda_3^2 - \lambda_1^2)(\lambda_3^2 - \lambda_2^2)},$$

[18] $a_m^{(2)} = a_{m-1}^{(2)} + m a_{m-1}^{(1)} = a_{m-1}^{(2)} + m$ である ($\because$ $a_{m-1}^{(1)} = 1$).

$\lambda = \lambda_1 = \lambda_2 = \lambda_3$ のとき

$$1 - e^{-\lambda T_{tact}} - \frac{e^{-\lambda T_{tact}}}{8}(5\lambda T_{tact} + \lambda^2 T_{tact}^2)\,,$$

$\lambda = \lambda_1 = \lambda_2 \neq \lambda_3$ のとき

$$1 + \frac{e^{-(\lambda+\lambda_3)T_{tact}}\{-2e^{\lambda T_{tact}}\lambda^4 + e^{\lambda_3 T_{tact}}\lambda_3^2(4\lambda^2 - 2\lambda_3^2 + \lambda^3 T_{tact} - \lambda\lambda_3^2 T_{tact})\}}{2(\lambda^2 - \lambda_3^2)^2}\,,$$

$\lambda = \lambda_2 = \lambda_3 \neq \lambda_1$ のとき

$$1 + \frac{e^{-(\lambda+\lambda_1)T_{tact}}\{-2e^{\lambda T_{tact}}\lambda^4 + e^{\lambda_1 T_{tact}}\lambda_1^2(4\lambda^2 - 2\lambda_1^2 + \lambda^3 T_{tact} - \lambda\lambda_1^2 T_{tact})\}}{2(\lambda^2 - \lambda_1^2)^2}\,,$$

$\lambda = \lambda_3 = \lambda_1 \neq \lambda_2$ のとき

$$1 + \frac{e^{-(\lambda+\lambda_2)T_{tact}}\{-2e^{\lambda T_{tact}}\lambda^4 + e^{\lambda_2 T_{tact}}\lambda_2^2(4\lambda^2 - 2\lambda_2^2 + \lambda^3 T_{tact} - \lambda\lambda_2^2 T_{tact})\}}{2(\lambda^2 - \lambda_2^2)^2}\,.$$

例題 11. 注意 2 中の係数 $a_m^{(j)}$ $(j = 1, 2, \ldots, m)$ のいくつかを下の表にまとめた．各 $a_m^{(j)}$ の値は 注意 2 で与えられた漸化式から求められる[19] ．

$a_1^{(1)} = 1$

$a_2^{(1)} = 1 \quad a_2^{(2)} = 2$

$a_3^{(1)} = 1 \quad a_3^{(2)} = 5 \quad a_3^{(3)} = 8$

$a_4^{(1)} = 1 \quad a_4^{(2)} = 9 \quad a_4^{(3)} = 33 \quad a_4^{(4)} = 48$

$a_5^{(1)} = 1 \quad a_5^{(2)} = 14 \quad a_5^{(3)} = 87 \quad a_5^{(4)} = 279 \quad a_5^{(5)} = 384$

$a_6^{(1)} = 1 \quad a_6^{(2)} = 20 \quad a_6^{(3)} = 185 \quad a_6^{(4)} = 975 \quad a_6^{(5)} = 2895 \quad a_6^{(6)} = 3840$

例えば，$m = 4$ のときは表の 4 行目の値を利用して，J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.15) ）は

$$1 - e^{-\lambda T_{tact}} - \frac{e^{-\lambda T_{tact}}}{48}(33\lambda T_{tact} + 9\lambda^2 T_{tact}^2 + \lambda^3 T_{tact}^3)$$

となる．$m = 5$ のときは表の 5 行目の値を利用して，J_i と J_{i+1} が衝突しない確率（すなわち，式 (3.15) ）は

$$1 - e^{-\lambda T_{tact}} - \frac{e^{-\lambda T_{tact}}}{384}(279\lambda T_{tact} + 87\lambda^2 T_{tact}^2 + 14\lambda^3 T_{tact}^3 + \lambda^4 T_{tact}^4)$$

となる．

[19] 参考：さらに求めると $a_7^{(1)} = 1$, $a_7^{(2)} = 27$, $a_7^{(3)} = 345$, $a_7^{(4)} = 2640$, $a_7^{(5)} = 12645$, $a_7^{(6)} = 35685$, $a_7^{(7)} = 46080$, $a_8^{(1)} = 1$, $a_8^{(2)} = 35$, $a_8^{(3)} = 588$, $a_8^{(4)} = 6090$, $a_8^{(5)} = 41685$, $a_8^{(6)} = 187425$, $a_8^{(7)} = 509985$, $a_8^{(8)} = 645120$ となる．

例題 12. 注意 2 中の係数 $a_m^{(j)}$ $(j = 1, 2, \ldots, m)$ の各関係式（漸化式）を解くと次のようになる（一部しかないが）[20] [21] .

$$\begin{aligned}
a_m^{(2)} &= \frac{1}{2}m^2 + \frac{1}{2}m - 1, \\
a_m^{(3)} &= 2 - \frac{7}{4}m - \frac{5}{8}m^2 + \frac{1}{4}m^3 + \frac{1}{8}m^4, \\
a_m^{(4)} &= -6 + 7m + \frac{5}{24}m^2 - \frac{17}{16}m^3 - \frac{11}{48}m^4 + \frac{1}{16}m^5 + \frac{1}{48}m^6 , \\
&\vdots
\end{aligned}$$

上で述べた解析結果を利用して，下に示した実用的な問題を考える．

実用的な問題. 衝突確率が与えられた定数 α $(0 \leq \alpha \leq 1)$ 以下になるという条件下で，タクトタイム T_{tact} を最小化せよ．

衝突確率は式 (3.17) で与えられる．それゆえ，次の不等式

$$1 - \left(\int_{-\infty}^{0} dy_1 \int_{-\infty}^{-y_1} dy_2 \cdots \int_{-\infty}^{-\sum_{i=1}^{m-1} y_i} g(y_1, y_2, \ldots, y_m) dy_m \right)^{n-1} \leq \alpha$$

をタクトタイムに関して解けばよい．実用的な問題を考えるときに注意するのは，式 (3.17) 中でタクトタイム T_{tact} のみが変数であり，そのほかの n, m, λ_j $(1 \leq j \leq m)$ は定数である．$m = 1$ のときは第 3.1.2 章で述べたように，上の不等式をタクトタイムに関して式変形で解くことができる．$m \geq 2$ では，式変形で解くのは難しい．以前にも述べたように，衝突確率はタクトタイムが短いほど高くなり，逆にタクトタイムを長くすれば長くするほど小さくなる．実際，式 (3.17) はタクトタイムに関して強い意味で単調減少である．そのため，タクトタイムに関して二分探索法を使って，最適な（最小の）タクトタイムを求めることができる．式 (3.17) は明示的に示されているので，たとえ m が大きな値だとしても，Mathematica などの科学技術計算ソフトウェアを使えば，容易に式 (3.17) の値を求めることができる．

[20] 漸化式を解くための詳細な手続きは省略.

[21] 一般の $a_m^{(j)}$ を式（例題中に示した格好）で表現するのは難しいと予想される.

第 4章

計算機シミュレーション

前章では，図 1.1 [1] に示した m 台の機械からなる直列型の搬送系を議論した．処理時間が正規分布に従うとき，衝突確率を与えたが，m の値が大きくなると衝突確率を具体的に計算して求めるのが困難になることを述べた．

しかし，本章で述べる計算機シミュレーションを実行することで，m の値が多くても衝突確率を求めることができる．図 1.1 では，ジョブが入口から搬送系へと投入されて，たくさんの機械で処理された後に，搬送系を退出する．理論的にはジョブを大量に搬送系へと投入すると，どこかの機械でジョブ同士の衝突が起きる確率が高まる．また，タクトタイムを短くすればするほど，衝突確率が高まる．機械の台数も衝突の有無に影響を与える．

本章では，このような振舞いを計算機を利用してシミュレーションをする方法，及びシミュレーションで得られた結果について考察する．具体的には，最初に各機械 M_j $(1 \leq j \leq m)$ でのジョブ J_i $(1 \leq i \leq n)$ の処理時間 $t_i^{(j)}$ に，擬似乱数を割り当てる．その後，擬似乱数を与えられた $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ を使って，衝突が起きるかどうかをチェックする．衝突の有無のチェックは補題 3 [2] を使って行うことができる．補題 3 より，衝突が存在するとき（M_j で衝突），かつそのときに限り，

$$\sum_{k=1}^{j} t_i^{(k)} > T_{\mathrm{tact}} + \sum_{k=1}^{j-1} t_{i+1}^{(k)}$$

を満たす i, j $(1 \leq i \leq n-1, 1 \leq j \leq m)$ の対が存在する．このことに基づいて，

[1] この図は第 1 章で示した．
[2] 補題 3 は第 2 章で示した．

衝突の有無をチェックするシミュレーションプログラムを作る.

4.1 処理時間が正規分布に従う場合

以下に，C 言語風に書いたシミュレーションプログラムを示す.

function *m-machine*
入力: $T_{\rm tact}$, n, m, μ_j, σ_j $(1 \le j \le m)$.
出力: 衝突が起きる機械のインデックス．ただし，衝突が生じない場合は 0.

```
1.   for (i = 1; i ≤ n; i++)
2.       for (j = 1; j ≤ m; j++)
3.           t_i^(j) ← N(μ_j, σ_j^2);    // t_i^(j) ~ N(μ_j, σ_j^2)
4.   for (i = 1; i ≤ n − 1; i++) {
5.       tmp_1 = t_i^(1);  tmp_2 = T_tact;
6.       if (tmp_1 > tmp_2) return 1;    // 機械 M_1 で衝突
7.       for (j = 2; j ≤ m; j++) {
8.           tmp_1 = tmp_1 + t_i^(j);
9.           tmp_2 = tmp_2 + t_{i+1}^(j−1);
10.          if (tmp_1 > tmp_2) return j;    // 機械 M_j で衝突
11.      }
12.  }
13.  return 0;    // 衝突無し
```

上の関数 m-machine では，処理時間が正規分布に従うと仮定している[3] ．n はジョブの個数，m は機械の台数，$T_{\rm tact}$ はタクトタイムをそれぞれ示す．機械 M_j における処理時間は正規分布 $\mathrm{N}(\mu_j, \sigma_j^2)$ に従うとする．1–3 行目で，全ての処理時間 $t_i^{(j)}$ を生成する．4–13 行目で，衝突の有無をチェックする．関数 m-machine は，機械 M_j で衝突する場合にはその機械のインデックス j を返し，衝突が存在しない場合には 0 を返す．一般にはいくつかの機械で衝突条件を満たしている場合もあるが，その場合でも 1 つのインデックスを返す.

[3]処理時間が指数分布など，他の分布に従う場合でも容易にシミュレーションできる．関数 m-machine の 3 行目を変更するだけである.

関数 m-machine の役割で特に重要なことは衝突の有無のチェックすることである．それゆえ，関数が 0 を返すかそれ以外を返すかに注目している．その意味で 0 以外の値が返ってくるのであれば，どれでもよい．衝突のチェックに関して，最初に J_1 と J_2 がどこかの機械 M_j $(1 \leq j \leq m)$ で衝突しているかどうかをチェックする．次に J_2 と J_3 がどこかの機械で衝突しているかどうかをチェックする．次に J_3 と J_4 に関しても同様に行い，最後には J_{n-1} と J_n のチェックを行う．

計算時間は，二重ループの構造を持つので，明らかに $\Theta(mn)$ である．また，メモリに関して処理時間のために二次元配列 $t_i^{(j)}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ を準備しているので，メモリ計算量は $\Theta(mn)$ である．

衝突の有無をチェックするのにいつも全ての $t_i^{(j)}$ を記憶しておく必要はない．例えば，J_1 と J_2 の衝突の有無をチェックするのに，J_i $(3 \leq i \leq n)$ の機械 M_j $(1 \leq j \leq m)$ での処理時間は必要ないからである．それゆえ，ジョブ二つ分の機械 M_j $(1 \leq j \leq m)$ での処理時間を蓄えておけば十分である．この考え方に基づいて修正したのが，関数 m-machine-1 である．実は巧妙なしかけにより，メモリに関してサイズ m の配列 A を使って実現している．関数 m-machine-1 のメモリ計算量は $\Theta(m)$ である．

function *m-machine-1*
入力: T_{tact}, n, m, μ_j, σ_j $(1 \leq j \leq m)$.
出力: 衝突が起きる機械のインデックス．ただし，衝突が生じない場合は 0.

```
1.  for (j = 1; j ≤ m; j++)
2.      A[j] ← N(μ_j, σ_j^2);    // A[j] ~ N(μ_j, σ_j^2)
3.  for (i = 1; i ≤ n − 1; i++) {
4.      tmp_1 = A[1];  tmp_2 = T_tact;
5.      if (tmp_1 > tmp_2) return 1;    // 機械 M_1 で衝突
6.      A[1] ← N(μ_1, σ_1^2);    // A[1] ~ N(μ_1, σ_1^2)
7.      for(j = 2; j ≤ m; j++) {
8.          tmp_1 = tmp_1 + A[j];
9.          A[j] ← N(μ_j, σ_j^2);    // A[j] ~ N(μ_j, σ_j^2)
10.         tmp_2 = tmp_2 + A[j − 1];
11.         if (tmp_1 > tmp_2) return j;    // 機械 M_j で衝突
12.     }
```

13. }
14. **return** 0; // 衝突無し

また，$n < m$ のときは関数 m-machine-2 を使えばよいだろう．配列 A のサイズをより小さくすることができるからである．ただし，メモリ計算量は $\Theta(m)$ で改善されない．なぜなら，入力された μ_j, σ_j $(1 \leq j \leq m)$ を蓄えるのに $\Theta(m)$ だけ必要になるからである．

function *m-machine-2*
入力: T_{tact}, n, m, μ_j, σ_j $(1 \leq j \leq m)$.
出力: 衝突が起きる機械のインデックス．ただし，衝突が生じない場合は 0.

1. **for** $(i = 1; i \leq n-1; i++)$ {
2. $A[i] \leftarrow N(\mu_1, \sigma_1^2)$; // $A[i] \sim N(\mu_1, \sigma_1^2)$
3. **if** $(A[i] > T_{\mathrm{tact}})$ **return** 1; // 機械 M_1 で衝突
4. }
5. $A[n] \leftarrow N(\mu_1, \sigma_1^2)$; // $A[n] \sim N(\mu_1, \sigma_1^2)$
6. **for** $(j = 2; j \leq m; j++)$ {
7. **for** $(i = 1; i \leq n-1; i++)$ {
8. $tmp \leftarrow N(\mu_j, \sigma_j^2)$; // $tmp \sim N(\mu_j, \sigma_j^2)$
9. $A[i] = A[i] + tmp$;
10. **if** $(A[i] > T_{\mathrm{tact}} + A[i+1])$ **return** j; // 機械 M_j で衝突
11. }
12. $tmp \leftarrow N(\mu_j, \sigma_j^2)$; // $tmp \sim N(\mu_j, \sigma_j^2)$
13. $A[n] = A[n] + tmp$;
14. }
15. **return** 0; // 衝突無し

関数 m-machine-1 と m-machine-2 の最悪時の時間計算量は両方とも $\Theta(mn)$ である[4]．こうして，衝突の有無のチェックは，$O(mn)$ 時間計算量，$\Theta(m)$ メモリ計算量で実装することができる．

[4]単に計算時間と書いた場合は，$O(mn)$ である．

さて，関数 m-machine を使って，衝突確率を算出することを考える．関数 m-machine を呼び出す回数を $LOOP$ とする．また，配列 $crash[0,\ldots,m]$ を用意する．関数 m-machine を $LOOP$ 回だけ呼び出したとき，$crash[j]$ は機械 M_j で衝突が生じた回数を意味する．ただし，$crash[0]$ は衝突が起きなかった回数を意味する．$crash[j]$ $(1 \leq j \leq m)$ を求める手続きを以下に示す．

for $(k = 1; k \leq LOOP; k++)$ {
　$index \leftarrow$ m-machine;　// 関数 m-machine を呼び出す．
　$crash[index]++$;
}

また，衝突が生じた回数は

$$\text{衝突が生じた回数} = \sum_{j=1}^{m} crash[j]$$

であるから，衝突確率は

$$\text{衝突確率} = \frac{\text{衝突が生じた回数}}{LOOP} \tag{4.1}$$

となる．

我々はシミュレーションを通して，$LOOP = 10,000$ に設定した．また，ジョブの個数 $n = 1{,}000$, $\mu_j = 1$, $\sigma_j = 0.01$ $(1 \leq j \leq m)$ に設定した．そのため，全ての機械での処理時間の分布は同一である．正規分布を $\mathrm{N}(1, 0.01^2)$ に設定したのは，実際の状況をかなりよく反映している[5]．平均値を 1 分と考えると，3 シグマ範囲[6] では 58.2 秒から 61.8 秒の区間になるが，この区間は実際的にも適当である．また，実際の製造装置のランニングテストでは，オブジェクトを 1,000 枚程度[7] 投入して，問題が無いかチェックする．そのため，$n = 1{,}000$ と設定するのは実際の状況を反映しており，妥当であると考える．擬似乱数生成については，LEDA[35], [36], [33], [1] のライブラリ関数を利用した．このような設定の下で衝突確率（式 (4.1)）を求める．

[5]三塚氏（大日本スクリーン株式会社）によると，“平均 1（分)、標準偏差 0.01（分） は、現状では妥当” とのこと．

[6]“事実上のすべて” の意味で，区間 $[\mu - 3\sigma, \mu + 3\sigma]$ を 3 シグマ範囲という [6].

[7]三塚氏によると，最低でも 500 枚くらいはランニングテストを行うとのこと．

表 4.1: 1 機械モデルでの衝突確率.

T_{tact}	1.02	1.03	1.04	1.05
シミュレーションで求めた値 (%)	100	72.98	2.79	0.07
理論値 (%)	100.00	74.06	3.11	0.03

$m = 1$ のときのシミュレーション結果を表 4.1 に示す．1 行目はタクトタイムを示しており，その範囲は 1.02 から 1.05 まで，0.01 刻みでシミュレーションを行った．2 行目は実際の結果（式 (4.1) を計算して求めたもの）を示している．$T_{\mathrm{tact}} = 1.02$ のときは衝突確率は 100% であった．T_{tact} を 1.02 よりも小さくしても，衝突確率は 100% のままである．T_{tact} を増やしていくと，徐々に衝突確率は低くなり，$T_{\mathrm{tact}} = 1.05$ のときに衝突確率は 0.07% で，ほとんど衝突が起きなかった．T_{tact} を 1.05 よりも大きな値に設定しても衝突確率はほぼ 0% になる．こうして予想通り，タクトタイムが増えると，衝突確率が低くなるという傾向を確認した．表 4.1 の 3 行目は衝突確率の理論値である．3 行目のそれぞれの値は第 3 章で与えた式 (3.3) を計算したものである．ただし，理論値の結果は全て，小数第 3 位を四捨五入したものである．表 4.1 から分かるように，シミュレーションによる値と理論値がほとんど同じになることを確認した．

次に $m = 2$ のときの結果を表 4.2 に示す．2 行目がシミュレーションによる衝突確率の結果である．1 機械モデルのときの結果と同様に，タクトタイムの増加とともに，衝突確率が低くなっていくのを確認することができる．

T_{tact} が 1.04 のときに 100% を示した．一方，1 機械モデルの場合には，同じタクトタイム（すなわち，1.04）のときに衝突確率が 2.79 % （表 4.1 を参照）しかなかった．このように，同じタクトタイムに設定しても，機械の台数が増えると衝突確率が増えることも確認することができる．その理由は，2 機械モデルの最初の機械 M_1 が 1 機械モデルの機械と同じ振舞いをするからである．1 機械モデルでの衝突確率が 2.79 % であるから，2 機械モデルでの M_1 でも同程度の衝突確率を示す．2 機械モデルの M_1 と M_2 の両方を考慮した衝突確率は 100% なので，衝突の大半は M_2 で生じていることが分かる．

表 4.2: 2 機械モデルでの衝突確率.

T_{tact}	1.04	1.05	1.06	1.07	1.08	1.09
シミュレーションで求めた値 (%)	100	85.78	22.89	2.58	0.19	0.01
理論値 (%)	100.00	85.72	23.34	2.62	0.19	0.01

表 4.2 の 3 行目は衝突確率の理論値である．3 行目のそれぞれの値は第 3 章で与えた式 (3.8) を計算したものである．表 4.2 から分かるように，シミュレーションによる値と理論値がほとんど同じになることを確認した.

次に $m \geq 3$ のときのシミュレーションによる衝突確率の結果を表 4.3 に示す. 各行は機械の台数が同じときの結果で，各列はタクトタイムが同じときの結果である．表 4.3 から，タクトタイムが増えるに従って衝突確率が低くなる傾向を確認できる．さらに，機械の台数 m が増えるに従って，衝突確率が高くなる傾向も確認できる.

また，m の値をさらに大きくした場合（$m = 10, 50, 100, 200, 400, 800, 1600$）のシミュレーション結果をそれぞれ表 4.4, 4.5 , 4.6, 4.7, 4.8, 4.9, 4.10 に示す．これらの表で，タクトタイムの区間については，だいたい衝突確率が 100% から 0 % まで減少していく部分を中心にチェックできるようにしている．1600 機械モデルでは表 4.10 から分かるように，他のモデルと比べて，タクトタイムをかなり大きくしないと衝突確率が低くならないことが分かる．例えば，衝突確率を 1% 未満にしたいなら，タクトタイムを 3.5 に設定しなければならない.

また，衝突確率の理論値は式 (3.17) で与えられるが，$m = 3, 4$ のときに計算した結果を表 4.11 に示す[8] ．表 4.3 と表 4.11 から，$m = 3, 4$ のときはシミュレーションで求めた衝突確率と理論式で求めた値とがほぼ一致することが分かる．m の値が 5 以上になると，式 (3.17) のロバストな計算はかなり難しい．例えば，Mathematica を使って計算しても，計算誤差（丸め誤差，数値積分による誤差など）のせいで，まったく見当違いの値が返ってくる.

[8]Mathematica で計算した.

表 4.3: シミュレーションで求めた m 機械モデルでの衝突確率 ($m = 3, 4, \ldots, 8$). [単位: %]

		T_{tact}						
		1.06	1.07	1.08	1.09	1.10	1.11	1.12
m	3	97.65	58.59	15.54	2.63	0.26	0.03	0
	4	100	98.79	72.27	28.79	7.32	1.52	0.24
	5	100	100	98.78	78.26	37.07	12.30	3.40
	6	100	100	100	97.74	76.54	39.63	15.02
	7	100	100	100	99.96	96.02	73.16	38.48
	8	100	100	100	100	99.77	93.56	68.51

		T_{tact}						
		1.13	1.14	1.15	1.16	1.17	1.18	1.19
m	3	0	0	0	0	0	0	0
	4	0.01	0.01	0	0	0	0	0
	5	0.76	0.10	0.01	0	0	0	0
	6	4.43	1.29	0.23	0.08	0.02	0	0
	7	15.38	5.41	1.52	0.40	0.12	0.04	0.01
	8	36.80	15.90	5.71	1.76	0.42	0.12	0.04

4.2 処理時間が指数分布に従う場合

処理時間が指数分布に従う場合にもシミュレーションを行った．このシミュレーションを行うためには，前に出てきた関数 m-machine の乱数生成の部分（3 行目）を変更するだけでよい．それ以外は，処理時間が正規分布に従う場合と全く同じようにやればよい．

乱数生成に関して，$\lambda_j = 1$ $(1 \leq j \leq m)$ の指数分布に設定した．また，ジョブ数 $n = 1,000$ とした．シミュレーションによる衝突確率の結果を表 4.12 に示す．表 4.12 から分かるように，正規分布と同じような傾向を示した．すなわち，タクトタイムの増加とともに衝突確率は減少し，機械の台数の増加とともに衝突確率

表 4.4: シミュレーションによる 10 機械モデルでの衝突確率 [単位: %]

T_{tact}	1.10	1.12	1.14	1.16	1.18	1.20
衝突確率 (%)	100	97.66	54.18	12.73	2.11	0.22

表 4.5: シミュレーションによる 50 機械モデルでの衝突確率 [単位: %]

T_{tact}	1.2	1.3	1.4	1.5	1.6
衝突確率 (%)	100	86.25	3.8	0.02	0

表 4.6: シミュレーションによる 100 機械モデルでの衝突確率 [単位: %]

T_{tact}	1.4	1.5	1.6	1.7
衝突確率 (%)	97.9	25.64	1.4	0.04

表 4.7: シミュレーションによる 200 機械モデルでの衝突確率 [単位: %]

T_{tact}	1.5	1.6	1.7	1.8	1.9	2.0
衝突確率 (%)	100	89.79	31.2	4.81	0.54	0.08

表 4.8: シミュレーションによる 400 機械モデルでの衝突確率 [単位: %]

T_{tact}	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4
衝突確率 (%)	100	98.54	72.53	29.23	8.09	1.75	0.33	0.04

表 4.9: シミュレーションによる 800 機械モデルでの衝突確率 [単位: %]

T_{tact}	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
衝突確率 (%)	99.64	92.33	65.93	35.78	16.03	5.92	1.94	0.64	0.21

表 4.10: シミュレーションによる 1600 機械モデルでの衝突確率 [単位: %]

T_{tact}	2.5	2.7	2.9	3.1	3.3	3.5
衝突確率 (%)	99.93	92.00	51.63	17.24	4.55	0.9

表 4.11: m $(=3,4)$ 機械での衝突確率の理論値. [単位: %]

		T_{tact}									
		1.06	1.07	1.08	1.09	1.10	1.11	1.12	1.13	1.14	1.15
m	3	97.65	58.63	15.96	2.79	0.39	0.04	0.01	0.00	0.00	0.00
	4	99.99	98.76	73.04	29.16	7.68	1.63	-	-	-	-

は増加する.

また，衝突確率の理論値を表 4.13 に示した．処理時間が指数分布に従うときの理論値は，式 (3.17) と 注意 2 の結果を利用して求めることができる．処理時間が正規分布に従うときと違って，m の値が 5 以上になっても容易に衝突確率を求めることができる．これは，衝突確率が明示的に表されているからである．表 4.12 と表 4.13 から，シミュレーションからの値と理論式から計算した値がほぼ一致することを確認できる.

4.3 2 機械並列モデル

直列型モデルの拡張としては，並列モデルが考えられる．一般の待ち行列理論では一つのステージにいくつもの窓口を仮定するのは普通なので，並列モデルは一般的な考え方である．ここでは，2 機械並列モデル（図 4.1 を参照）に関して，シミュレーションを行った結果を述べる.

以下のことを仮定している.

- 入口から投入されたジョブは，M_1 が空いていれば，M_1 で処理を開始する.
- M_1 が別のジョブを処理中で，M_2 が空いていれば，M_2 で処理を開始する.

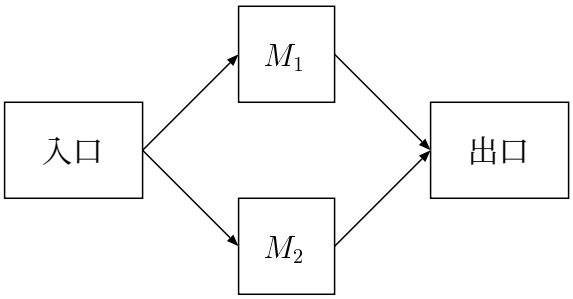


図 4.1: 2 台の同種の処理機械からなる搬送系.

- M_1 も M_2 も別のジョブを処理中ならば，衝突が生じる．

実験条件は，$n = 1{,}000$, $\mu_j = 1$, $\sigma_j = 0.01$ $(1 \leq j \leq m)$ として，以下のプログラムを走らせた．

```
ft1 = 0;    ft2 = 0;
ft1 = pt[1];
for(i=2; i<=N; i++) {
    if(ft1 <= (i-1)*Tact) {    // M1 が空かどうかのチェック
        ft1 = (i-1)*Tact + pt[i];
    } else if(ft2 <= (i-1)*Tact) {    // M2 が空かどうかのチェック
        ft2 = (i-1)*Tact + pt[i];
    } else {
        return 1;    // 衝突
    }
}
return 0;
```

ft1 が機械 M_1 での処理終了時間を，ft2 が M_2 での処理終了時間を意味している．また，1 を返すとき衝突を意味しており，0 を返すときは，衝突が生じなかったことを意味する．上のプログラムを 10,000 回実行して，どの程度衝突が起きたかを調べた．その結果を表 4.14 に示す．この結果から，1 機械モデルでの衝突確率と同じ値を実現するのに，ほぼ半分のタクトタイムでよいことが分かった（表 4.1 を参照）．

4.4 バッファの効果

ここでは，バッファ付き m 機械からなる直列型搬送系（図 1.2 を参照）を議論する．実際の製造装置は通常，バッファを用意しているので，バッファ付きのモ

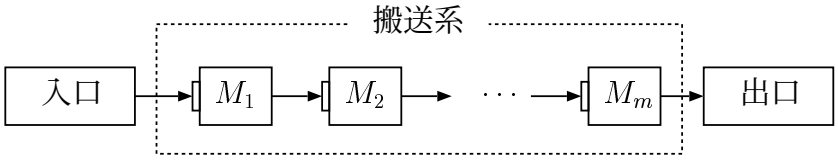


図 4.2: 各機械が一つ分のバッファを持つときの直列型搬送系.

デルを議論するのは自然なことである．実際，[28] では有限個のバッファをどの機械にどの程度置けばよいかを研究している．この論文 [28] では，スループット[9] を目的関数としている．一方，我々の目的関数は衝突確率である．衝突確率を低くするには，どのようなバッファ配置が良いかを考える．バッファ有りの場合，衝突確率を解析的に求めるのはかなり難しいと予想しているが，計算機シミュレーションから様々な有益な知見を得ることができる．

バッファ無しの搬送系（図 1.1 を参照）では，ジョブ J_i $(2 \leq j \leq n)$ が M_j $(1 \leq j \leq m)$ に到着したときに，M_j が一つ前に投入されたジョブ J_{i-1} を処理中であれば，J_{i-1} と J_i の衝突が生じる．しかし，M_j が一つのバッファを持っていれば，衝突を回避することができる．なぜなら，J_i はそのバッファへと一時的に退避することができるからである．そして，J_{i-1} の処理が完了するとすぐに，M_j はバッファにあるジョブ J_i を自動的に取り出し処理を開始する．ここで，簡単のためにバッファへジョブを置いたり，あるいはバッファから取り出しに要する時間は無いと仮定する．実際はこの時間を無視することはできないが，その場合は処理時間がそのような時間も含んでいるとみなせば，ここでの議論を適用することができる．

各機械がバッファを一つ持つ搬送系（図 4.2 を参照）では，ジョブ J_i $(3 \leq i \leq n)$ が M_j $(1 \leq j \leq m)$ に到着したとき，M_j と M_j のバッファがそれぞれ J_{i-2} と J_{i-1} で占有されているときに，衝突が生じる．

ここでは，バッファありモデルとバッファなしモデルの二つの場合を，衝突確率に関してシミュレーションを利用して，比較する．シミュレーション結果は，各機械がバッファを一つ持つだけでも劇的に衝突確率が低くなった．説明を簡単にするために，以下の変数を導入する．

[9] 処理装置が，単位時間当たりに処理できるジョブの個数.

- $st_i^{(j)}$: M_j が J_i の処理を開始する時刻
- $ft_i^{(j)}$: M_j が J_i の処理を完了する時刻

ただし，すべての $i\,(1 \le i \le n)$ に対して，$ft_i^{(0)} = (i-1)\cdot T_{\rm tact}$ とする．各機械がバッファを一つ持つとき，衝突に関して次の観察を見ることができる．

観察 1. $i, j(2 \le i \le n-1, 1 \le j \le m)$ に対して，機械 M_j でジョブ J_i と J_{i+1} が衝突するための必要十分条件は $st_i^{(j)} > ft_{i+1}^{(j-1)}$ である．

上の観察から分かるように，ジョブ J_1 と J_2 の衝突は起こらない．上の観察に基づいて，各機械が一つのバッファを持つとき，衝突の有無をチェックするシミュレーションプログラムを以下に示す．プログラム m-machine-buffer は C 言語風に書かれている．

function *m-machine-buffer*
入力: $T_{\rm tact}$, n, m, μ_j, σ_j $(1 \le j \le m)$.
出力: 衝突が生じる機械のインデックス．ただし，どの機械でも衝突が生じない場合は 0.

```
1.  for (i = 1; i ≤ n; i++)
2.      for (j = 1; j ≤ m; j++)
3.          t_i^(j) ← N(μ_j, σ_j^2);    // t_i^(j) ~ N(μ_j, σ_j^2)
4.  for (j = 0; j ≤ m; j++) ft_0^(j) = 0;
5.  for (i = 1; i ≤ n; i++) ft_i^(0) = (i-1)·T_tact;
6.  for (i = 1; i ≤ n; i++)
7.      for (j = 1; j ≤ m; j++) {
8.          st_i^(j) = max{ft_i^(j-1), ft_{i-1}^(j)};
9.          ft_i^(j) = st_i^(j) + t_i^(j);
10.     }
11. for (i = 2; i ≤ n-1; i++)
12.     for (j = 1; j ≤ m; j++)
13.         if (st_i^(j) > ft_{i+1}^(j-1)) return j;
14.          // ジョブ J_i と J_{i+1} が 機械 M_j （のバッファ）で衝突.
15. return 0; // 衝突が生じない
```

関数 m-machine-buffer では，処理時間が正規分布に従うと仮定している．n はジョブの個数，m は機械の台数，T_{tact} はタクトタイムをそれぞれ示す．機械 M_j における処理時間は正規分布 $\mathrm{N}(\mu_j, \sigma_j^2)$ に従うとする．1–3 行目で，全ての処理時間 $t_i^{(j)}$ を生成する．残りの行で，衝突の有無をチェックする．関数 m-machine-buffer は，機械 M_j で衝突する場合にはその機械のインデックス j を返し，衝突が存在しない場合には 0 を返す．一般にはいくつかの機械で衝突条件を満たしている場合もあるが，その場合でも 1 つのインデックスを返す．関数 m-machine の役割で特に重要なことは衝突の有無のチェックすることである．それゆえ，関数が 0 を返すかそれ以外を返すかに注目している．その意味で 0 以外の値が返ってくるのであれば，どれでもよい．計算時間は，二重ループの構造を持つので，明らかに $\Theta(mn)$ である．

関数 m-machine-buffer を $10,000$ 回実行して，そのうち何回衝突が生じて，何回衝突が生じなかったかを数えた．ジョブの個数 $n = 1,000$, $\mu_j = 1$, $\sigma = 0.01$ $(1 \leq j \leq m)$ に設定した．そのため，すべての機械での処理時間の分布は全く同一である．また，タクトタイムは 0.997 から 1.001 まで，0.001 間隔で設定してシミュレーションを行った．表 4.15 は計算機実験の結果をまとめたものである．

表 4.15 から分かるように，衝突確率は各機械に一つのバッファを持つだけで，バッファ無しの場合と比べて，劇的に低くなった．タクトタイムが 1.001 のときに衝突確率は 0% である．一方，バッファ無しのモデルでは同じタクトタイムで衝突確率は 100% であった．

また，タクトタイムを各機械の処理時間の平均値と同じ値（すなわち，1.000）に設定したときでも，衝突確率はかなり低く，表 4.15 からほぼ 0 になる．タクトタイムを各機械の処理時間の平均値と同じ値に設定することは，全体の処理時間最小化（あるいは，処理機械の稼働率最大化）を目指す場合の目標であると考えているので，得られた結果は実用的に非常に役立つものと考える．

さらに，機械の台数が増えてもほとんど衝突確率が増加しないことが分かった．1 機械からはじめて 800 機械でもほとんど同じ衝突確率を示した．バッファなしの搬送系では，全体の衝突確率は機械の台数の増加とともに高くなる傾向があったが，この場合にはそのような傾向は見られない．この性質は実用的にもかなり役立つものと考えている．

表 4.15 の結果から，タクトタイムが 0.997, 0.998, 0.999, 1.000 のときに衝突確率は 0 にならなかった．これらの確率を 0 にするにはどうしたらいいだろうか．手っ取り早い方法は，単にバッファの数を増やせばよい．ただ，バッファをたくさん準備するとその分コストがかかるので，できるだけバッファを減らすように現場の要求がある．そのため，無駄にバッファを設けるのではなく，必要な分だけ（衝突確率減少に貢献する分だけ）用意しなくてはならない．では，どの機械にどの程度のバッファを持たせるといいだろうか．それを調べるために，仮にすべての機械がバッファを無限個持っているとしよう．そうすることで衝突は絶対に生じない．そして，各機械で必要なバッファ数を数える．以下に，シミュレーションプログラムを示す．

```
// 正規分布に従う処理時間を生成
for(j=1; j<=M; j++){
    for(i=1; i<=N; i++){
        pt[i][j] = nrnd() * Sigma + Mu; // 乱数生成
        if(pt[i][j] < 0) pt[i][j] = 0.0;
    }
}

// M1 におけるスケジュールを求める．
st[1][1]=0;
ft[1][1]=st[1][1]+pt[1][1];
for(i=2; i<=N; i++) {
    st[i][1] = max2(ft[i-1][1], (i-1)*Tact);
    ft[i][1] = st[i][1] + pt[i][1];
}

// J1 のスケジュールを求める．
for(j=2; j<=M; j++){
    st[1][j] = ft[1][j-1];
    ft[1][j] = st[1][j] + pt[1][j];
}

// J2 - J_N のスケジュールを求める．
for(i=2; i<=N; i++) {
    for(j=2; j<=M; j++) {
        st[i][j] = max2(ft[i-1][j], ft[i][j-1]);
        ft[i][j] = st[i][j] + pt[i][j];
    }
}

// M1 で必要なバッファの個数を求める．
for(i=2; i<=N; i++) {
    buffer = 0;
```

```
        for(k=1; k<=i-1; k++) {
            if(ft[k][1] > (i-1)*Tact) {
                buffer = i-k;
                break;
            }
        }
        if(buffer > maxbuffer[1]) maxbuffer[1] = buffer;
    }

    // M2 - M_M で必要なバッファの個数を求める.
    for(j=2; j<=M; j++) {
        for(i=2; i<=N; i++) {
            buffer = 0;
            for(k=1; k<=i-1; k++) {
                if(ft[k][j] > ft[i][j-1]) {
                    buffer = i-k;
                    break;
                }
            }
            if(buffer > maxbuffer[j]) maxbuffer[j] = buffer;
        }
    }
```

上のプログラムで，maxbuffer[j] が機械 M_j で必要な（実際に利用された）バッファの個数を示している．実際の実験では，上のプログラムを $LOOP$ 回実行した．それゆえ各 j に対して，$LOOP$ 個の maxbuffer[j] を得ることになるが，そのうちの最大値を機械 M_j で必要なバッファ数とした．実験条件（$n = 1{,}000$, $\mu_j = 1$, $\sigma_j = 0.01$）は先のシミュレーションと同じである．ただし，$m = 800$ とする．$LOOP = 100$ のときの実験結果（M_j で必要なバッファ数）を以下に示す．

$T_{\text{tact}} = 0.997$ のとき　$M_1 : 4, \quad M_2 : 2, \quad M_3 \sim M_{800} : 1.$

$T_{\text{tact}} = 0.998$ のとき　$M_1 : 3, \quad M_2 : 2, \quad M_3 \sim M_{800} : 1.$

$T_{\text{tact}} = 0.999$ のとき　$M_1 : 2, \quad M_2 : 2, \quad M_3 \sim M_{800} : 1.$

$T_{\text{tact}} = 1.000$ のとき　$M_1 : 2, \quad M_2 \sim M_{800} : 1.$

$T_{\text{tact}} = 1.001$ のとき　$M_1 \sim M_{800} : 1.$

上の結果から分かるように，M_1 や M_2 のような最初の方に置かれている機械が，残りの機械と比較して，多くのバッファを必要とすることが分かった．機械のインデックスが増えるに従って，必要なバッファ数は単調に減少している．こ

の実験から，どの機械にどのくらいのバッファが必要か知ることができて，無駄なバッファを減らすことができる．

次に $LOOP = 10,000$ のときの結果を以下に示す．

$T_{\mathrm{tact}} = 0.997$ のとき　$M_1 : 5, \quad M_2 \sim M_5 : 2, \quad M_6 : 1, \quad M_7 : 2, \quad M_8 \sim M_{800} : 1.$

$T_{\mathrm{tact}} = 0.998$ のとき　$M_1 : 4, \quad M_2 \sim M_5 : 2, \quad M_6 : 1, \quad M_7 : 2, \quad M_8 \sim M_{800} : 1.$

$T_{\mathrm{tact}} = 0.999$ のとき　$M_1 : 3, \quad M_2 \sim M_5 : 2, \quad M_6 : 1, \quad M_7 : 2, \quad M_8 \sim M_{800} : 1.$

$T_{\mathrm{tact}} = 1.000$ のとき　$M_1 \sim M_5 : 2, \quad M_6 \sim M_{800} : 1.$

$T_{\mathrm{tact}} = 1.001$ のとき　$M_1 \sim M_{800} : 1.$

上の結果から分かるように，$LOOP$ が増えると，当然ながら必要なバッファ数も増える傾向を示した．ただ，必要なバッファ数は，機械のインデックスが増えるに従って，単調に減少しない部分があった（例えば，タクト長が 0.997 のとき，M_6 と M_7 のバッファ数を参照）．上の実験をすることでで主張できることは，“$LOOP$ 回のシミュレーションを行っても，このぐらいのバッファを用意しておけば，衝突が起きない”ということである．$LOOP$ の値が大きいほど，必要なバッファ数も増えて，そのシステムの安全性も向上するものと考えている．その意味で，$LOOP$ の値をどの程度にするのかが重要である．

参考までに，$LOOP = 100,000$ のときの結果[10] を以下に示す．

$T_{\mathrm{tact}} = 0.998$ のとき　$M_1 : 4, \quad M_2 : 3, \quad M_3 \sim M_{12} : 2, \quad M_{13} \sim M_{800} : 1.$

[10] この結果は三浦くんの提供によるものである．この結果を得るためにかかった CPU タイムは 16678.8 秒．

表 4.12: シミュレーションで求めた m 機械での衝突確率 $(m = 1, 2, \ldots, 8)$．ただし，処理時間は指数分布に従う．[単位: %]

		T_{tact}								
		2	3	4	5	6	7	8	9	10
m	1	100	100	100	99.91	91.89	60.05	28.49	11.96	4.92
	2	100	100	100	100	100	98.52	82.00	49.67	24.07
	3	100	100	100	100	100	100	99.06	87.64	58.78
	4	100	100	100	100	100	100	100	99.02	88.33
	5	100	100	100	100	100	100	100	99.98	98.75
	6	100	100	100	100	100	100	100	100	99.94
	7	100	100	100	100	100	100	100	100	100
	8	100	100	100	100	100	100	100	100	100

		T_{tact}								
		11	12	13	14	15	16	17	18	19
m	1	1.74	0.59	0.16	0.06	0.03	0.01	0.01	0.01	0
	2	10.87	4.29	1.64	0.71	0.36	0.18	0.08	0.05	0.01
	3	31.08	15.58	7.16	3.05	1.39	0.66	0.28	0.06	0.03
	4	62.98	36.09	18.50	8.57	3.93	1.61	0.68	0.26	0.10
	5	87.31	62.87	36.67	19.14	8.93	4.22	1.73	0.74	0.36
	6	97.34	84.08	60.14	35.75	19.46	9.15	4.30	1.87	0.86
	7	99.84	95.51	80.11	56.44	33.19	17.79	9.02	4.20	1.86
	8	100	99.25	93.00	75.61	51.57	30.20	16.32	8.00	3.91

表 4.13: m 機械での衝突確率の理論値 ($m = 1, 2, \ldots, 8$). ただし，処理時間は指数分布に従う．[単位: %]

		T_{tact}								
		2	3	4	5	6	7	8	9	10
m	1	100.00	100.00	100.00	99.88	91.62	59.80	28.48	11.60	4.43
	2	100.00	100.00	100.00	100.00	100.00	98.36	81.31	49.25	23.83
	3	100.00	100.00	100.00	100.00	100.00	100.00	99.09	87.35	59.19
	4	100.00	100.00	100.00	100.00	100.00	100.00	99.99	99.04	88.41
	5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	99.98	98.57
	6	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	99.93
	7	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	8	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

		T_{tact}								
		11	12	13	14	15	16	17	18	19
m	1	1.65	0.61	0.23	0.08	0.03	0.01	0.00	0.00	0.00
	2	10.28	4.21	1.68	0.66	0.26	0.10	0.04	0.02	0.01
	3	31.88	15.01	6.60	2.81	1.17	0.48	0.20	0.08	0.03
	4	62.64	35.83	17.91	8.32	3.71	1.62	0.70	0.30	0.13
	5	87.20	62.41	36.81	19.14	9.25	4.30	1.95	0.87	0.38
	6	97.57	84.42	59.95	35.82	19.10	9.52	4.56	2.13	0.98
	7	99.76	95.77	80.33	56.04	33.58	18.21	9.28	4.56	2.19
	8	99.99	99.29	92.89	75.07	51.24	30.64	16.79	8.72	4.37

表 4.14: 2 機械並列モデルでの衝突確率.

T_{tact}	0.51	0.515	0.52	0.525
衝突確率 (%)	100	72.97	2.79	0.07

表 4.15: シミュレーションによる衝突確率. [単位: %]

		T_{tact}				
		0.997	0.998	0.999	1.000	1.001
m	1	100	99.97	60.46	0.32	0
	2	100	99.99	64.64	0.68	0
	3	100	100	64.48	0.83	0
	4	100	99.99	64.26	0.88	0
	5	100	100	65.23	0.88	0
	6	100	100	64.40	0.99	0
	...	...	...	...	...	...
	50	100	100	66.43	0.96	0
	100	100	100	66.16	0.89	0
	200	100	100	65.70	0.89	0
	400	100	99.98	65.44	1.02	0
	800	100	99.99	65.00	0.98	0

第 5章

その他のスケジューリング問題

本章では，実際の製造の現場で重要なその他のスケジューリング問題を議論する．特に第 5.1 章では搬送計画問題を扱い，第 5.2 章では周期的なタイムスロット付きジャストインタイムスケジューリング問題を扱う．

5.1 搬送計画問題

搬送計画問題とは，多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．図 5.1 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装置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, 決められた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. 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

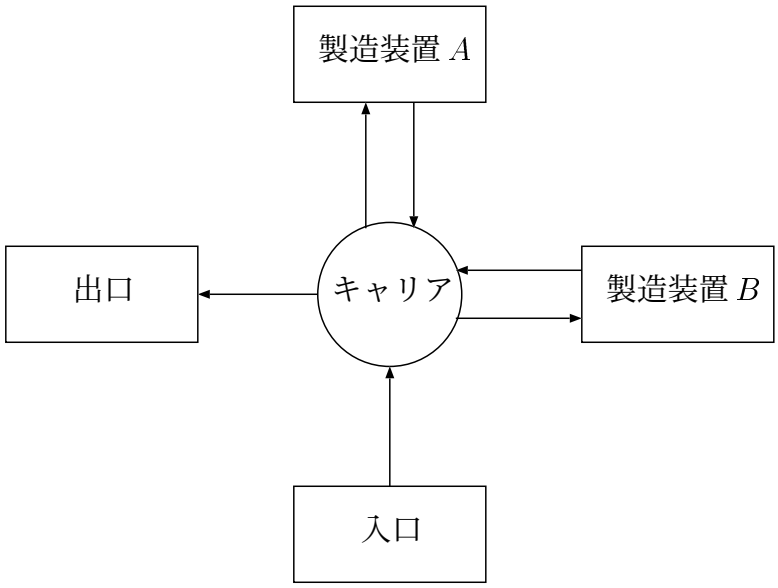


図 5.1: 簡単な製造装置の模式図

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 5.1 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 5.2 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

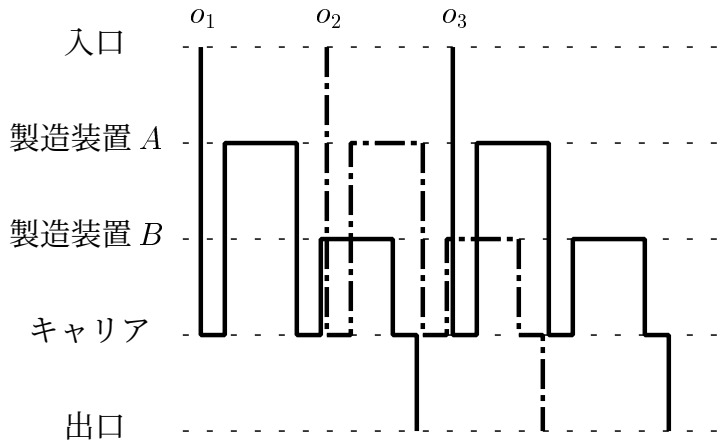


図 5.2: 搬送計画の例

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. 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 5.2 は満たしている. 図 5.2 以外にも実行可能な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（搬送スケジュール）である.

5.1.1 搬送計画問題の定義，及び例題

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.

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

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.

例題 13. 以下の入力を考える.

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

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. 上の入力に対して, 制約条件を満た

す搬送計画を図 5.3 (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 5.3 (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 5.3 (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 5.3 (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 5.3 (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 5.3 (c) は 1 だけずらした搬送計画なので最適である.

例題 14. 全く同じ移動系列を複数個処理する場合を考える.

$$P = \{p_1, p_2, p_3, p_4, p_5\}, O = \{o_1, o_2, o_3\},$$
$$\sigma(o_1) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$$
$$\sigma(o_2) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$$
$$\sigma(o_3) = (p_1[0,0], p_5[1,1], p_2[3,5], p_5[1,1], p_3[3,5], p_5[1,1], p_4[3,5], p_5[1,1], p_1[0,0]),$$
$$cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 1.$$

まず, 明らかに制約条件を満たす搬送計画を図 5.4 (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 5.4 (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. そのような例を下で考える.

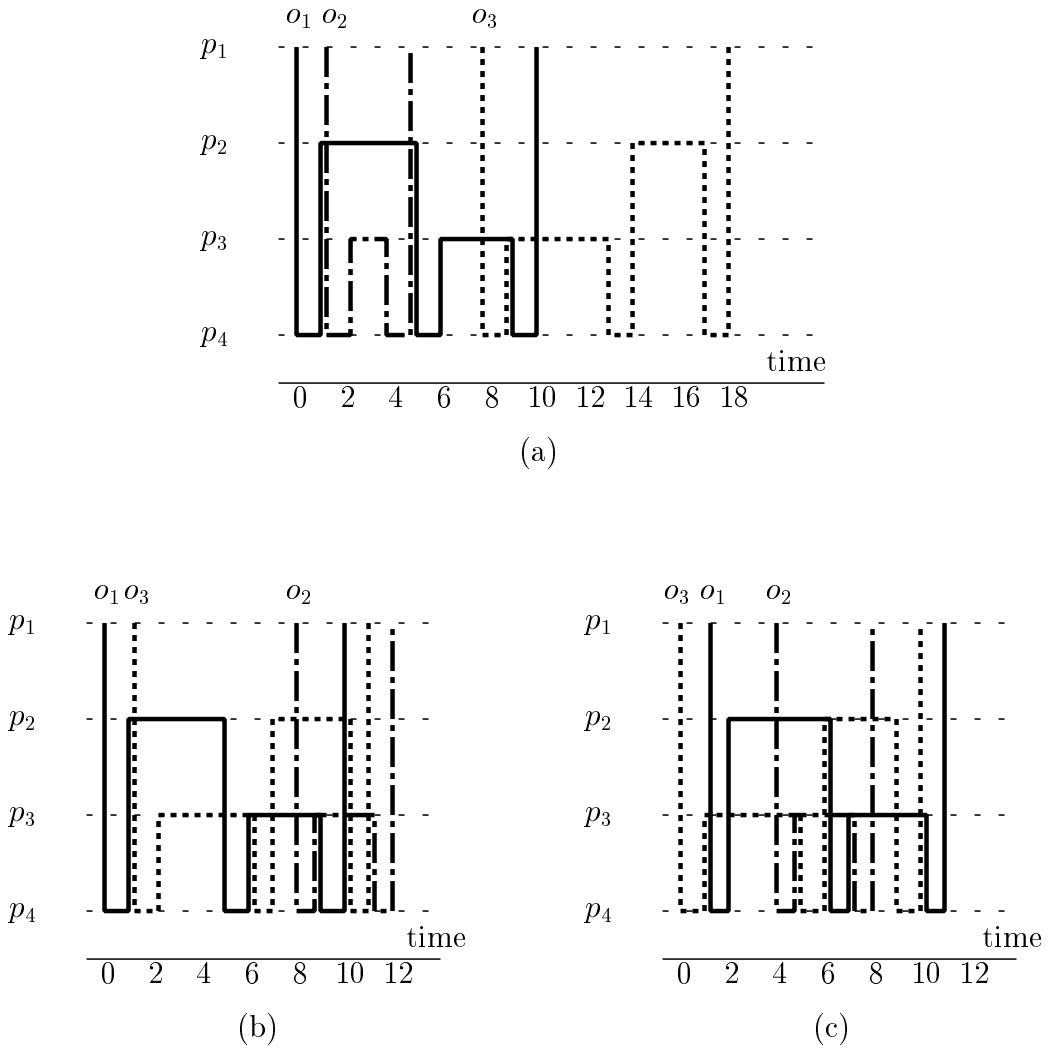


図 5.3: 搬送計画の例

例題 15. 以下の入力を考える.

$$P = \{p_1, p_2, p_3\}, O = \{o_1, o_2\},$$
$$\sigma(o_1) = (p_1[0,0], p_3[1,1], p_2[3,6], p_3[1,1], p_1[0,0]),$$
$$\sigma(o_2) = (p_1[0,0], p_3[5,5], p_1[0,0]),$$
$$cap(p_1) = cap(p_2) = cap(p_3) = 1.$$

明らかに制約条件を満たす搬送計画を図5.5 (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 5.5 (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

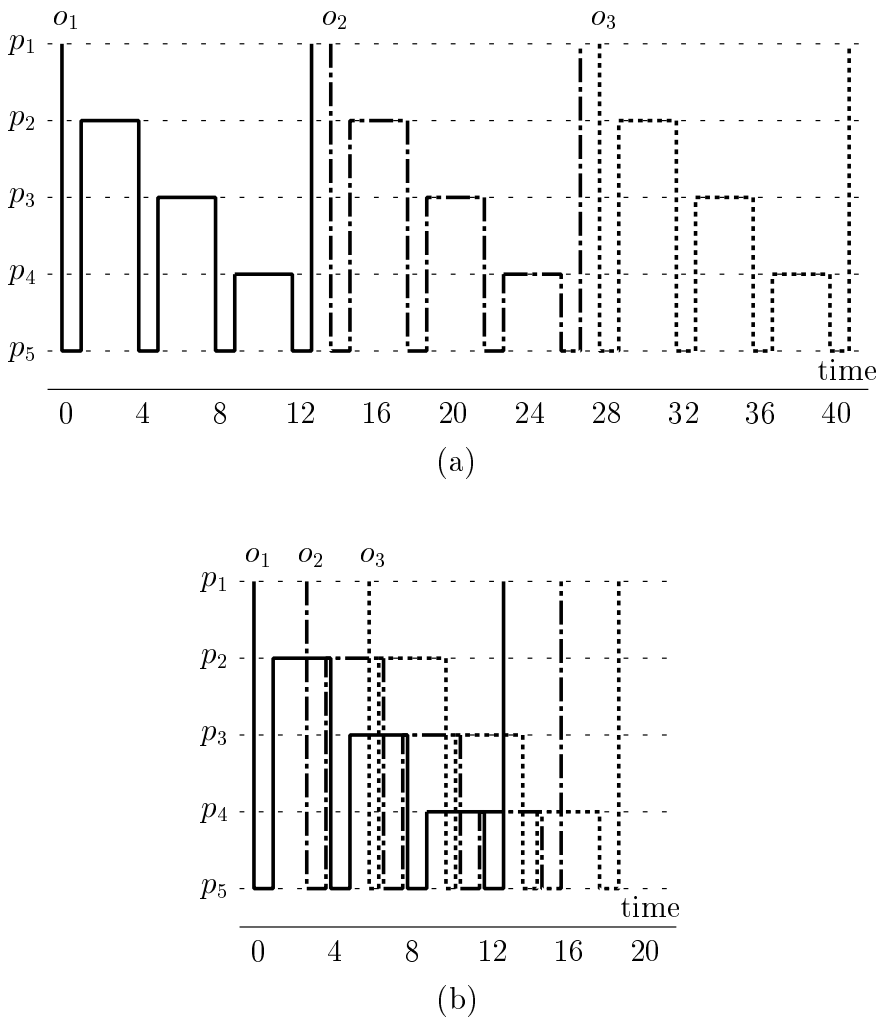


図 5.4: 搬送計画の例

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.

最適な搬送計画が複数個存在するケースを見てみる.

例題 16. 次の入力を考える.

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

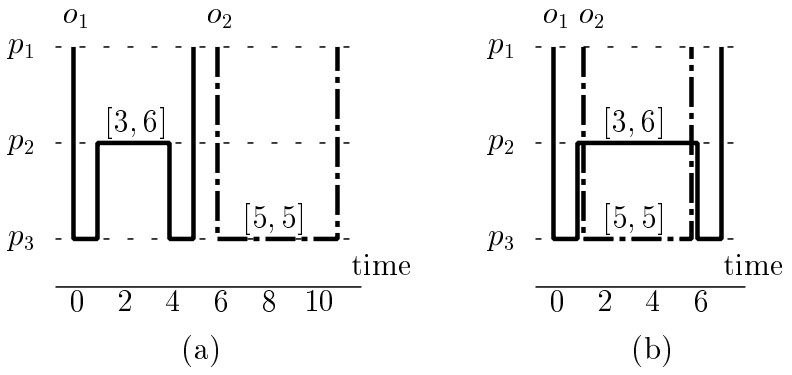


図 5.5: 搬送計画の例

明らかに制約条件を満たす搬送計画を図 5.6 (a) に示す. また, 図 5.6 (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. 3 つとも最適なスケジュールを示している.

これまで見てきた例題はすべての場所のキャパシティが 1 であった. そうでない例も見てみる.

例題 17. 以下の入力を考える.

$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}, O = \{o_1, o_2, o_3, o_4\},$

$\sigma(o_1) = (p_1[0,0], p_6[1,1], p_2[4,6], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_1[0,0]),$

$\sigma(o_2) = (p_1[0,0], p_6[1,1], p_2[4,6], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_1[0,0]),$

$\sigma(o_3) = (p_1[0,0], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_5[4,6], p_6[1,1], p_1[0,0]),$

$\sigma(o_3) = (p_1[0,0], p_6[1,1], p_3[4,6], p_6[1,1], p_4[4,6], p_6[1,1], p_5[4,6], p_6[1,1], p_1[0,0]),$

$cap(p_1) = cap(p_6) = 1, cap(p_2) = cap(p_3) = cap(p_4) = cap(p_5) = 2.$

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 5.7 (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対応する.

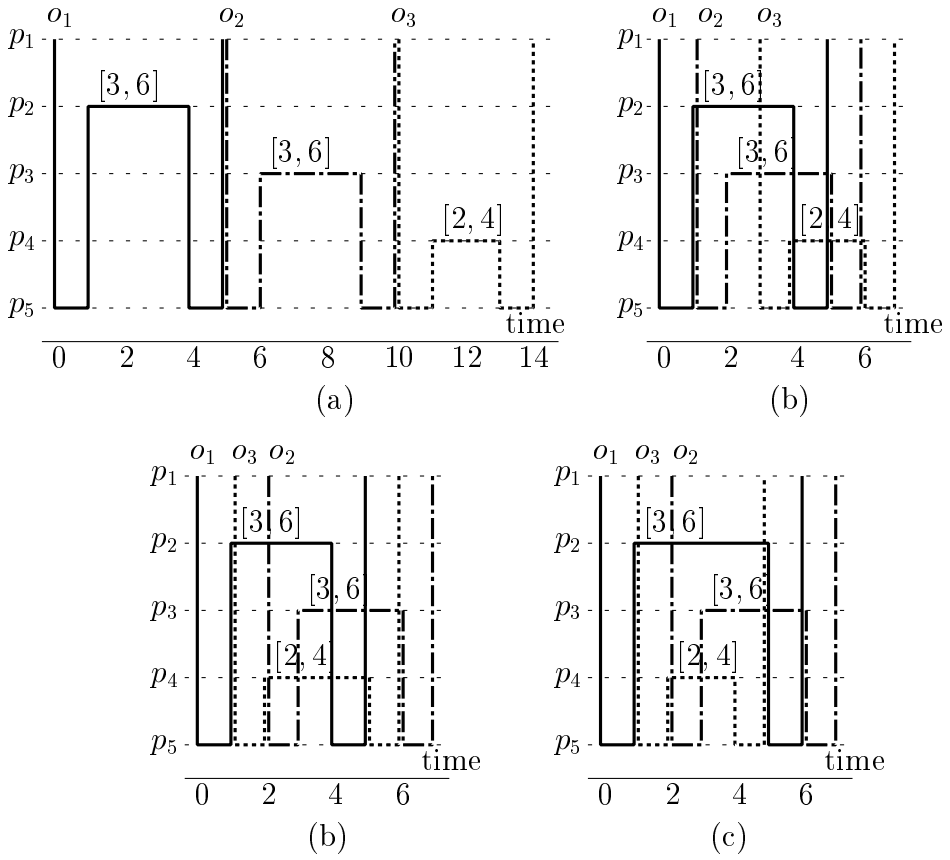


図 5.6: 搬送計画の例

5.1.2 *NP* 完全性

定理 1. 搬送計画問題は NP 完全である.

証明. 以下の二つのことを示せばよい.

- 搬送計画問題が NP に属すること.
- 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.

次に, ビンパッキング問題が搬送計画問題へ多項式時間還元可能であることを示す. ここで, ビンパッキング問題 (箱詰め問題とも呼ばれる) とは代表的な NP 完全問題の一つであり [39] , 次のような問題である.

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

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

$P = \{p_1, p_2\}, O = \{o_1, o_2, \cdots, o_n, o_{n+1}, \cdots, o_{n+k}\}$,

$\sigma(o_i) = (p_1[L_i, L_i])(1 \leq i \leq n)$,

$\sigma(o_i) = (p_1[1, 1], p_2[C, C], p_1[1, 1])(n + 1 \leq i \leq n + k)$,

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

$T = k(C + 2)$

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 5.8 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. (証明終)

定理 2. 各オブジェクトのスタートの順序が決まっていたとしても搬送計画問題はNP 完全である.

証明. 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（制約付き搬送計画問題）への入力 :

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

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 5.9 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. (証明終)

5.1.3 制約付き搬送計画問題に対する解法

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. ここで，イベントとはある場所 p_i $(\in P)$ から別の場所 $p_{i'}$ $(\in P \setminus \{p_i\})$ へと移動することを意味する．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.

また，ネットワーク理論を利用したアプローチもあり，より高速に解くことができる．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 [20]．この手法を，制約付き搬送計画問題に適用させる.

まずは，線形計画法に解法を述べる．例として，下の入力に対して搬送計画問題を考える.

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

この入力に対する制約条件を満たす搬送計画を図 5.10 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, 以下の線形計画問題を解けばよい.

$$
\begin{aligned}
\text{最小化} \quad & x_8 \\
\text{条件} \quad & d(1,2) = 1, \quad 3 \le d(2,3) \le 5, \quad d(3,4) = 1, \\
& d(5,6) = 1, \quad 3 \le d(6,7) \le 5, \quad d(7,8) = 1, \\
& d(3,6) \ge 0, \quad d(2,5) \ge 0, \quad d(4,7) \ge 0, \\
& x_2 = x_1 + d(1,2), \quad x_3 = x_2 + d(2,3), \quad x_4 = x_3 + d(3,4), \\
& x_6 = x_5 + d(5,6), \quad x_7 = x_6 + d(6,7), \quad x_8 = x_7 + d(7,8), \\
& x_6 = x_3 + d(3,6), \quad x_5 = x_2 + d(2,5), \quad x_7 = x_4 + d(4,7).
\end{aligned}
$$

次にネットワークフローを利用した解法を説明する. 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 5.10 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. 図 5.11 に図 5.10 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.

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.

$$
\begin{aligned}
\text{最小化} \quad & x_t - x_s \\
\text{条件} \quad & x_j - x_i \ge d(i,j) \quad \text{for each arc}(i,j) \in A
\end{aligned}
\tag{5.1}
$$

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.

例題 18. 以下の入力を考える.

$$
\begin{aligned}
& P = \{p_1, p_2, p_3, p_4\}, \quad O = \{o_1, o_2\}, \\
& \sigma(o_1) = (p_1[0,0], p_4[1,1], p_2[3,7], p_4[1,1], p_1[0,0]), \\
& \sigma(o_2) = (p_1[0,0], p_4[1,1], p_3[3,5], p_4[1,1], p_1[0,0]), \\
& cap(p_1) = cap(p_2) = cap(p_3) = cap(p_4) = 1.
\end{aligned}
$$

この入力に対する矛盾の無い搬送計画を図 5.12 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 5.13 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．

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, 式 (5.1) 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．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 [39]．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．図 5.13 の (G, d) から構成した $(G, \bar{d})$ を図 5.14 に示す.

$(G, \bar{d})$ 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 法を使えば, 一般に $O(mn) = O(n^2)$ で解くことができる [49]．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, 式 5.1 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 5.13 の (G, d) から構成した $(G, \tilde{d})$ を図 5.15 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．よって，入力サイズ n の制約付き搬送計画問題を $O(n \log n)$ で解けることが分かった.

5.2 周期的なタイムスロット付きジャストインタイムスケジューリング問題

長い間，スケジューリング問題における目的関数として，全体の処理完了時間（メイクスパン）や，メイクスパンに関して非減少な関数が考えられてきた．このような目的関数は正則であると呼ばれる．多くの文献は，平均フロー時間，平均遅れ時間，遅れたジョブの割合（各用語の定義は [26] , [4] などを参照）などの正則である目的関数を扱っている．最近では，ジャストインタイム（JIT）生産に対する要求が高まってきたので，正則ではない目的関数を扱った研究への期待も徐々に高まってきている [34]．JIT 生産において理想的なスケジュールとは，すべてのジョブが与えられた納期ちょうどに処理を完了するようなスケジュールである．実はそのような理想的なスケジュールを扱った研究は少ない．例えば，[53] の 16.1 章で，ある種の古典的で有名なスケジューリング問題を取り上げている．この問題は，活動選択問題（activity-selection problem）と呼ばれる．

最近，活動選択問題を自然に拡張した問題に対して，効率の良いアルゴリズムが開発された [45], [46]．また，上の問題よりもさらに一般化した問題に対して，多項式時間アルゴリズムが提案されている [31]．[45] では，平石ら [31] の論文の主な成果は納期付きで同種の並列機械モデルに対する研究結果である，と書いてある．[31] で扱った問題は次のようなものである．各ジョブは非負の重みを持つ．また，全てのジョブ間には非負のセットアップタイムが与えられる．そして，目的関数として，ジャストインタイムで処理されたジョブの重みの合計を最大化する．ここで，ジョブが与えられた納期ちょうどぴったりに処理を完了するならば，そのジョブはジャストインタイムで処理される，と呼ぶ．[31] では，ジャストインタイムで処理することができないジョブに関しては，どのように扱うかを考慮していない．そこで，本研究では次のような実際の製造状況を考える．ある周期の中で処理を出来なかったジョブは，次の周期で処理されるだろう（例えば，明日，来週，来月など）．我々はそのような状況を考慮して，周期の長さのことを“周期的なタイムスロット”と呼ぶ．

我々の問題では，すべてのジョブがジャストインタイムで処理されなくてはならないが，その理由は製品のストック代をできるだけ減らしたいという要求から

生じている．例えば，製品が予定の出荷日よりも早く完成してしまうと，その分だけ余計にストック代がかかってしまう．また，周期的なタイムスロットという概念は，実際の製造に関する状況から生じた．例えば，出荷日はある一定の間隔（毎日，毎週，毎月など.）に固定されていることが多い．我々の問題は，最初に平石ら [30] によって研究された．そこでは，同種の並列機械モデルを扱っている．我々の研究のモチベーションは，多くの現実的な問題が存在することである．例えば，自動車の製造や人工衛星による観察などが挙げられる．

以下では，周期的なタイムスロット付きジャストインタイムスケジューリング問題を議論する．各ジョブには，処理時間と周期的に繰返される納期が割り当てられる．ただし，すべてのジョブに対して，周期は同じとする．セットアップタイムはすべてのジョブの順序対に対して与えられる．目的関数は，与えられたすべてのジョブを処理するのに必要な各機械での周期的なタイムスロット数のうち，それらの最大値を最小化することである．ただし，各ジョブは周期的な納期のうち，どこかでちょうどぴったりに処理が完了される．我々はまず，扱う問題をある種の最適化問題として定式化する．次に，$P \neq NP$ の仮定の下で，多項式時間で計算可能な任意の関数 $\alpha(n)$ に対して，$\alpha(n)$ 以内の近似率で近似することは不可能であることを証明する．次に，一機械モデルの場合にネットワークフローを利用したあるヒューリスティックアルゴリズムを提案する．さらに，計算機シミュレーションの結果を示す．最後に，我々のヒューリスティックアルゴリズムが最適解を返す場合や，近似比 1.5 を達成する場合を示す．また，我々のヒューリスティックアルゴリズムは，セットアップタイムの上界に依存して近似比を与えることを示す．

5.2.1 問題の定義

我々の問題を定義するために以下の変数を準備する．ただし，正整数 k に対して，集合 $I_k = \{1, 2, \cdots, k\}$ とする．

- M_i $(i \in I_m)$: m 個の同種機械
- J_j $(j \in I_n)$: n 個のジョブ
- $p_j > 0$ $(j \in I_n)$: 各ジョブの処理時間

- $d_j \geqq p_j$ $(j \in I_n)$: 各ジョブの納期
- $s_{jk} \geqq 0$ $(j \neq k \wedge j,k \in I_n)$: J_j と J_k 間のセットアップタイム
- $L \geqq \max_{j \in I_n}\{d_j\}$: タイムスロット長

タイムスロットは周期的であり，それぞれのタイムスロット中にジョブの納期が与えられる．すなわち，ジョブ J_j の納期は $d_j, L+d_j, 2L+d_j, \cdots$ であり，これらの納期中のどれか一つの時点で J_j の処理が完了する．また，ジョブ間には先行関係が無く，並列に処理される．そして，各機械で要するタイムスロット数の最大値を最小にするスケジュールを求める問題を考えよう．

ジョブ J_j のスケジュールとは写像 $S : J_j \mapsto (M^S_{[j]}, C^S_j)$ である．ただし，$M^S_{[j]}$ はジョブ J_j が処理される機械であり，C^S_j はジョブ J_j の処理が終了する時刻である．そして，以下の 2 つの条件を満たす時，スケジュール S は実行可能であると言う．

(i) 各ジョブ J_j に対して，制約式 $C^S_j = r^S_j \cdot L + d_j$ を満たす非負の整数 r^S_j が存在する．

(ii) $M^S_{[j]} = M^S_{[k]} (j \neq k \wedge j,k \in I_n)$ であり，J_j と J_k が連続して処理されるならば，次の 2 つの制約式 $C^S_k \geqq C^S_j + s_{jk} + p_k$, $C^S_j \geqq C^S_k + s_{kj} + p_j$ のどちらか一方が成り立つ．

また，$r(S) := \max_{j \in I_n}\{r^S_j\}$ とすると，周期的なタイムスロット付きジャストインタイムスケジューリング問題を次のように問題を記述できる．

入力: ジョブ数 n，機械数 m，処理時間 p_j，納期 d_j，セットアップタイム s_{jk}，タイムスロット長 L

目的関数: $r(S)$ →最小

制約条件: スケジュール S は実行可能である

以下，前後関係から明らかな時は，単に問題と省略して呼ぶことにする．全ての実行可能なスケジュールの中で，$r(S)$ が最小になるスケジュールを最適解と呼び，その時の $r(S)$ を最適コストと呼ぶ．各機械で必要なタイムスロット数の最大値は $r(S)+1$ である．$m \geqq n$ の時は，明らかにどの機械も二つ以上のジョブを処理し

表 5.1: 入力例

j	p_j	d_j
1	5	7
2	2	10
3	5	16
4	2	3
5	1	5
6	7	13

ないスケジュールが存在するので，簡単に解ける．そのため，以下では $m < n$ と仮定する．また，上で各納期 $d_j \geqq p_j$ と仮定したのは，全てのジョブは一つのタイムスロット内で処理されることに対応する．

ここで，異なる 2 つのジョブ J_j, J_k に対して，制約式 $(g_{jk} - 1) \cdot L + d_k < d_j + s_{jk} + p_k \leqq g_{jk} \cdot L + d_k$ を満たす非負整数 g_{jk} を導入する．この制約式は，J_j を処理した後，g_{jk} タイムスロット先で J_k の処理を完了することを意味する．問題の入力が与えられた時，任意の $j, k (j \neq k \wedge j, k \in I_n)$ に対して g_{jk} が一意に定まる．ここで，$g := \max_{j \neq k \wedge j,k \in I_n} \{g_{jk}\}$ とする．先行研究 [47] から，以下の結果が知られている．

- 問題は $m = 1$ の時でさえ，*NP*-困難である．
- $g \leqq 1$ ならば，問題は多項式時間で解ける．

例題 19. 表 5.1 に示した入力を考える．ジョブ数 $n = 6$，機械数 $m = 1$，セットアップタイム $s_{jk} = 0 (j \neq k \wedge j, k \in I_6)$，タイムスロット長 $L = 17$.

最初のタイムスロットから順次，[*31*] の方法を用いて可能な限り多くのジョブを処理していくというグリーディな手法を考えよう．この場合，図 5.16 に示す様に，全体のスケジュールはタイムスロットを 3 つ必要とする．しかし，図 5.17 に示す様に，明らかにより少ないタイムスロット数（= 2 個）で全体の処理を完了するスケジュールが存在する．

5.2.2 近似不可能性

定理 3. 多項式時間で計算可能な任意の関数 $\alpha(n)$ に対して，P = NP でない限り，タイムスロット付きジャストインタイムスケジューリング問題を $\alpha(n)$ 以内の近似率で近似することは不可能である．

証明: 背理法を用いる[1] ．タイムスロット付きジャストインタイムスケジューリング問題に対して，近似率 $\alpha(n)$ の多項式時間近似アルゴリズム $\mathcal{A}$ が存在したとする．すると，$\mathcal{A}$ が *NP*-困難なハミルトンパスの存在判定問題 (例えば，[39], [41] などを参照) を多項式時間で解くのに使えてしまい，*P=NP* となってしまうことを以下に示す．

まず，ハミルトンパスの存在判定問題からタイムスロット付きジャストインタイムスケジューリング問題への還元を説明する．ハミルトンパスの存在判定問題は次の様に定義される．

入力: 有向グラフ $G = (V, E)$．ただし，$V = \{v_1, \cdots, v_n\}$ とする．

出力: ハミルトンパス (各頂点をちょうど一度通る有向パス) が存在するか？

これに対して，タイムスロット付きジャストインタイムスケジューリング問題の入力を以下の様に構成する．

入力: ジョブ数 n，機械数 = 1，タイムスロット長 = 1，各納期 $d_j = 1$，各処理時間 $p_j = 1$，
$(v_i, v_j) \in E$ ならば，$s_{ij} = 0$，そうでないなら，$s_{ij} = \alpha(n) \cdot n$

すると，以下の性質を満たす．

- G がハミルトンパスを持つならば，タイムスロット付きジャストインタイムスケジューリング問題の最適解のタイムスロット数は n であり，
- G がハミルトンパスを持たないならば，タイムスロット付きジャストインタイムスケジューリング問題の最適解のタイムスロット数は $\alpha(n) \cdot n$ より大きい．

[1] ここでの証明法は，[54] の 3.2 章に書いていることを参考にした．

タイムスロット付きジャストインタイムスケジューリング問題に対して，近似率 $\alpha(n)$ のアルゴリズム $\mathcal{A}$ を走らせると，最初のケースに当てはまるときはタイムスロット数が $\alpha(n) \cdot n$ 以下の解を返し，第 2 のケースに当てはまる時はスロット数が $\alpha(n) \cdot n$ より大きい解を返す．したがって，$\mathcal{A}$ は G がハミルトンパスを持つかどうか判定するのに使える． (証明終)

5.2.3 ヒューリスティックアルゴリズム

本章では，ネットワークフローを利用したヒューリスティックアルゴリズムを記述する．問題の入力が与えられた時，以下に示すような単純な重み付き連結有向グラフ $G = (V, A)$ を構成する．ここで，キャパシティ関数を $c : A \to \mathbb{N}$，重み関数を $w : A \to \mathbb{Z}$ とする．

- $V = \{s, t, a_i, b_i \mid i \in I_n\}$ であり，$2n + 2$ 個の節点から成る．
- A は以下に示す $n^2 + 2n$ 個の枝から成る．
  - $w(s, a_j) = 0$ を付した (s, a_j) $(j \in I_n)$
  - $w(a_j, b_j) = -w_{\text{job}}$ を付した (a_j, b_j) $(j \in I_n)$
  - $w(b_j, t) = 0$ を付した (b_j, t) $(j \in I_n)$
  - $w(b_j, a_k) = g_{jk}$ を付した (b_j, a_k) $(j \neq k \wedge j, k \in I_n)$

ただし，節点 s の入次数は 0 で，ソースと呼ばれる．節点 t の出次数は 0 で，シンクと呼ばれる．また，w_{job} は，制約式 $g < w_{\text{job}}$ を満たす任意の正整数である．点 a_j, b_j $(j \in I_n)$ はジョブ J_j に対応する．

例題 20. 表 5.2 に示した入力を考える．ジョブ数 $n = 4$，機械数 $m = 1$，セットアップタイム $s_{jk} = 1$ $(j \neq k \wedge j, k \in I_4)$，タイムスロット長 $L = 8$.

上に示した様な入力が与えられた時，10 個の要素から成る節点集合と 24 個の要素から成る枝集合の対から成るグラフ G を構成する．また，それぞれ $g_{12} = 0, g_{13} = 1, g_{14} = 0, g_{21} = 1, g_{23} = 1, g_{24} = 1, g_{31} = 1, g_{32} = 1, g_{34} = 0, g_{41} = 2, g_{42} = 1, g_{43} = 1$ であることを考慮して，構成される G を図 5.18 に示す．

表 5.2: 入力例

j	p_j	d_j
1	2	2
2	2	6
3	3	4
4	2	8

さて，各枝 $(u, v) \in A$ の容量 $c(u, v)$ を 1 とし，単位流量当りの費用 $w(u, v)$ が与えられているときに，1 点 s から他の 1 点 t への流量 m の整数流 $f : A \to [0, 1]$ の中で総費用を最小にするもの（s から t への流量 m の最小費用流）を求める問題を考える．数式で表せば，次の様になる．

『条件

$$\sum_w f(v, w) - \sum_u f(u, v) = \begin{cases} m & (v = s), \\ 0 & (v \in V \setminus \{s, t\}), \\ -m & (v = t), \end{cases}$$

$$f(u, v) \in [0, 1] \text{ for any } (u, v) \in A$$

の下で

$$w_f = \sum_{(u,v)\in A} w(u, v) f(u, v)$$

を最小にせよ.』

構成したグラフ G に対して，最小費用流を求める．この最小費用流は，各枝に対して，0 か 1 のどちらか一方の値を決めている．全ての枝 (a_j, b_j) にはフロー値 1 が対応するが，その理由は，枝の重み $-w_{\text{job}}$ が制約式 $g < w_{\text{job}}$ を満たしているからである．ここで，フロー値が 1 の枝をグラフに残し，フロー値が 0 の枝をグラフから削除したグラフを G' とおく．G' には m 個のソースからシンクへのパスが存在することになる．もし，G の全ての枝コストが正であるならば，G' には m 個のソースからシンクへのパス以外に，パスは存在しない．しかし，G には枝コストに負のものがあるので，G' には m 個のソースからシンクへのパス以外に，いくつかのサイクルが存在することもあり得る．このようなサイクルが生じる例を

図 5.19 に示す．この例は，図 5.18 のグラフ G に対して最小費用流を求め，グラフ G' を表したものである．もし，G' においてサイクルが生じなければ，次の様にして G' を実行可能解へ対応させる．G' においてソースからシンクへのパスの個数は流量 m に等しいが，これらの m 個のパスをフロー $f_1, f_2, \cdots, f_m$ で表す．すなわち，フロー f_i は，対応するパス上の枝に対して値 1 を与え，そうでない枝に対して値 0 を与える．すると，各フロー f_i を 1 台の機械へ対応させることが出来る．$\{f_1, f_2, \cdots, f_m\}$ から $\{M_1, M_2, \cdots, M_m\}$ への写像は全部で $m!$ 個あるが，機械は同種なので，どの写像を選んでもよい．また，フロー f_i に対応するパス π は，正の重みを持つ全ての枝を取り除くことによって，複数のパス $\pi_1, \pi_2, \cdots, \pi_l$ に分解できる．各パス π_i $(i = 1, 2, \cdots, l)$ に以下の様にタイムスロットを割り当てる．

(i) π_1 に最初のタイムスロットを割り当てる．

(ii) π_j に k 番目のタイムスロットを割り当てるとする．この時，π_j と π_{j+1} を接続する枝の重みが k' であるならば，π_{j+1} に $(k + k')$ 番目のタイムスロットを割り当てる．

以上が G' においてサイクルが生じない場合の実行可能解への対応の取り方である．また，その時の「延べ使用タイムスロット数」は

$$m + \sum_{(u,v) \in A \ \wedge \ w(u,v) > 0} w(u, v) f(u, v)$$

である．最小費用流問題の最適解は上の「延べ使用タイムスロット数」を最小化するが,「各機械で要するタイムスロット数の最大値」を最小化するわけではない．ただし，機械数 $m = 1$ であれば両者は等しくなるので，最小費用流問題の最適解が対応するスケジュールは本問題の最適解でもある．ここで問題となるのは G' にサイクルが存在するときである．この時，G' は実行可能なスケジュールに対応していない．

以下，機械数 $m = 1$ の場合に限定して議論する．G' にサイクルが存在する時でも，次の様にして実行可能解を得る．

ヒューリスティックアルゴリズム:

1. グラフ G を構成する．

2. G に対して，最小費用流を求め，G' にサイクルが存在しなければ，終了．

3. G' のサイクル上の全ての枝 (b_k, a_j) に対して，$\min\{w(b_k, a_i) - w(b_k, a_j), w(b_l, a_j) - w(b_k, a_j)\}$ が最小である枝 (b_k, a_j) を求める．ただし，ソースからシンクへのパス上で，a_i はソースの次の節点であり，b_l はシンクの直前の節点である．

4. ステップ 3 で求めた枝 (b_k, a_j) に対して，
   IF $w(b_k, a_i) - w(b_k, a_j) < w(b_l, a_j) - w(b_k, a_j)$

   G' において，枝 (b_k, a_j)，(s, a_i) を削除，枝 (b_k, a_i)，(s, a_j) を付け加える．

   ELSE

   G' において，枝 (b_l, t)，(b_k, a_j) を削除，枝 (b_l, a_j)，(b_k, t) を付け加える．

5. G' にサイクルが存在しなければ，終了．あれば，ステップ 3 へ．

ステップ 4 は枝の付け替え処理であり，その様子を図 5.20 に示す．図 5.20(a) は枝の付け替え前に対応している．図 5.20(b) は，ステップ 4 の IF 文で条件式が成り立つ時の枝の付け替えに対応している．図 5.20(c) は，ステップ 4 の IF 文で条件式が成り立たない時の枝の付け替えに対応している．このヒューリスティックアルゴリズムにより，最終的に G' にはサイクルが存在しないので，実行可能解へ対応させることが出来る．また，その時のタイムスロット数は

$$1 + \sum_{(u,v)\in A \ \wedge \ w(u,v)>0} w(u,v) f(u,v)$$

である．

実際，ヒューリスティックアルゴリズムのステップ 2 で最小費用流の計算を行う前に，負の重みを持った枝を取り除くために，arc reversal transformation (cf. [50]) を行う．そうすることで，フロー値が $n+1$ になる．なぜなら，$n+1$ のうち，1 つ分がもともとのフロー値であり，残りの n の分が arc reversal transformation によって増えた分だからである．最小費用流は，[50] に書かれている successive shortest path algorithm を使うと，$O(n^3)$ の計算時間で求めることができる．ヒューリスティックアルゴリズム全体の計算時間も支配しているのは，明らかに最小費用流

の計算である．それゆえ，我々のアルゴリズムは $O(n^3)$ 時間である実行可能スケジュールを返す．

注意 3. 上ではネットワークフローアルゴリズムを利用した手法を提案したが，巡回セールスマン問題（*TSP*）とサイクルカバーを利用して同じ問題を解くこともできる．具体的に言えば，グラフ G を構成して *1* 機械の場合の我々の問題を，*TSP* へと還元する．そして，最小コストサイクルカバー問題（すなわち，与えられた有向グラフ上のすべての点をカバーする有向サイクルの集合のうち最小コストをもつものを求める問題）の解を利用する．最小コスト *cycle cover* 問題は *TSP* の緩和問題であり，よく知られている割り当て問題への還元を利用することで，$O(n^3)$ 時間で解くことができる *[16]*．この問題に対する最適解が一つのサイクルから構成されるならば，それは最適な *TSP* ツアーでもある．しかし，一般には最適解は複数のサイクルから構成される．我々はそのようなサイクルをつなげる．

5.2.4 計算機実験

実験環境を以下に示す．

Machine: Dell Precision 650

CPU: Intel Xeon 3.06GHz × 2

OS: Microsoft Windows 2003 Server

Memory: 4GB

Compiler: Microsoft Visual C++ 6.0

前章で提案したヒューリスティックアルゴリズムの性能を評価するために，アルゴリズムを実装して上に示した環境で計算機実験を行った．実装の際，C++のクラスライブラリである LEDA を用いてプログラムを作成した [35]．特に最小費用流の計算は，LEDA で用意されている関数 MIN_COST_FLOW() を利用した．また，機械数 $m = 1$，タイムスロット長 $L = 20$ の下で，以下に示すランダムな入力データを対象とした．

- $0 <$ 各納期 $d_j \leqq L$

ジョブ数	5	10	20	40	80	160	320
ステップ 2 で終了する割合 (%)	35	21	5	4	5	2	0

表 5.3: ヒューリスティックアルゴリズムがステップ 2 で終了する割合

ジョブ数	最大値	平均値	分散	理論的な最大値
100	8	3.56	2.4264	49
200	10	4.08	3.1136	99
400	12	4.57	4.8051	199
800	12	5.14	4.3804	399
1600	11	5.8	4.46	799

表 5.4: ヒューリスティックアルゴリズムのステップ 2 におけるサイクルの個数

- $0 <$ 各処理時間 $p_j \leqq d_j$
- $0 \leqq$ 各セットアップタイム $s_{jk} \leqq L$

ヒューリスティックアルゴリズムのステップ 2 で G' にサイクルが存在しなければ，そこで終了して最適解を得ることになるが，100 回の試行を行いその割合を調べた．実験結果を表 5.3 に示す．各行はそれぞれ入力されるジョブの個数，及びヒューリスティックアルゴリズムのステップ 2 で終了する割合を表している．表 5.3 から分かるように，ジョブの個数が比較的少ない時は最適解を得る確率が高いが，逆にジョブの個数が 320 では，100%の確率でサイクルを生じる．

それでは，どのくらいの個数のサイクルが生じているのであろうか．そのことを調べるために，100 回の試行を行った結果を表 5.4 に示す．各列はそれぞれ入力されるジョブの個数，得られたサイクル数の最大値，サイクル数の平均，分散，そして理論的なサイクル数の最大値を表している．ここで，理論的なサイクル数の最大値は，ジョブ数 n に対して，$\lfloor \frac{n-1}{2} \rfloor$ である．表 5.4 から分かるように，理論的な最大値と実験による最大値に大きな開きを確認した．また，平均値はジョブの個数が増えるに従ってわずかに増加するが，理論的な最大値の増加量に比べると少ない．

次に，ステップ 2 でサイクルを生じた場合，ヒューリスティックアルゴリズムから求まるタイムスロット数と全探索アルゴリズムから求めた最適なタイムスロッ

ジョブ数	近似比の最悪値	近似比の平均	分散
3	1	1	0
4	1.5	1.00978	0.00300402
5	1.33333	1.01522	0.00366418
6	1.5	1.01932	0.00393428
7	1.4	1.0224	0.00393347
8	1.4	1.02461	0.00370951
9	1.4	1.0237	0.00326279
10	1.4	1.02431	0.00301326

表 5.5: 計算機実験によるヒューリスティックアルゴリズムの近似比

ト数との近似比を調べた．10000 回の試行を行った結果を表5.5 に示す．各列はそれぞれ入力されるジョブの個数，得られた近似比の最悪値，近似比の平均，そして分散を表している．表 5.5 から分かるように近似比の最悪値と比べて，平均値の方はほぼ 1 であり良好な結果となった．分散についてもかなり小さな値となった．

最後に，ヒューリスティックアルゴリズムの計算時間を LEDA で用意された関数 used_time() を用いて計測した結果を図 5.21 に示す．ジョブ数が 100 , 200 , 300 , $\cdots$,2000 のそれぞれに対して 100 回の試行を行い平均を取った．図 5.21 で実線は最小費用流を求める（ヒューリスティックアルゴリズムのステップ 2) のに要する計算時間で，点線は枝の付け替え処理も含めた（ヒューリスティックアルゴリズムのステップ 2,3,4,5）計算時間を示している．図 5.21 から分かるように，計算時間は最小費用流を求めることが支配的であり，高速である．

5.2.5 近似比

ここでは，近似比に関する結果を述べる．ヒューリスティックアルゴリズムのステップ 1 で構成したグラフ G に対して，以下の変数を導入する．

$$\delta_{k*} := \max_{k \neq j \ \wedge \ j \in I_n} \{w(b_k, a_j)\} - \min_{k \neq j \ \wedge \ j \in I_n} \{w(b_k, a_j)\}$$

$$\delta_{*j} := \max_{j \neq k \ \wedge \ k \in I_n} \{w(b_k, a_j)\} - \min_{j \neq k \ \wedge \ k \in I_n} \{w(b_k, a_j)\}$$

補題 6. $\delta := \max_{k \neq j \wedge j,k \in I_n} \min\{\delta_{k*}, \delta_{*j}\}$，$\gamma^*$ が最適スケジュールに必要なタイムスロット数であるとすると，ヒューリスティックアルゴリズムで求められるスケジュールのタイムスロット数は高々

$$\gamma^* + \delta \cdot \left\lfloor \frac{n-1}{2} \right\rfloor \tag{5.2}$$

である．

証明. 入力されるジョブの個数を n とする．ヒューリスティックアルゴリズムのステップ 2 で求められる G' において，サイクルの個数は高々$\lfloor \frac{n-1}{2} \rfloor$ であり，ヒューリスティックアルゴリズムのステップ $3, 4, 5$ のループ回数に等しい．また，枝の付け替えに伴うコスト増加量は高々δであることを考慮して主張が導かれる．(証明終)

次にある仮定の下で近似比を導出する．

定理 4. ヒューリスティックアルゴリズムのステップ 1 で得られるグラフ G について，$w(b_j, a_k) \in \{1, 2\}$（ただし，$j \neq k \wedge j, k \in I_n$）である時，ヒューリスティックアルゴリズムは近似比 1.5 を保証する．

証明: γ^* を最適スケジュールに必要なタイムスロット数とする．また，γ_f^* を最小費用流から求まるタイムスロット数とする．この時，$\gamma_f^* \leqq \gamma^*$ が成り立つ．また，入力となるジョブの個数を n とすると，$w(b_j, a_k) \in \{1, 2\}$ より，n は γ^* の下界である．すなわち，$n \leqq \gamma^*$ が成り立つ．以上から，γ をヒューリスティックアルゴリズムから求められるタイムスロット数とすると，

$$\gamma \leqq \gamma_f^* + \left\lfloor \frac{n-1}{2} \right\rfloor \leqq \gamma^* + \frac{n-1}{2}$$

が成り立つので，以下の関係式が導かれる．

$$\frac{\gamma}{\gamma^*} \leqq 1 + \frac{(n-1)/2}{\gamma^*} \leqq 1 + \frac{(n-1)/2}{n} < 1.5$$

よって，近似比 1.5 が導かれた． (証明終)

次に，一般の場合の近似比に関して議論する．そのために，次の変数

$$w_j^* = \min_{k \in I_n} \{w(b_j, a_k)\},$$

$$w^* = \max_{j \in I_n} \{w_j^*\},$$

$$c = \frac{(\sum_{j=1}^{n} w_j^*) - w^*}{n-1}$$

を導入する．すると，$c(n-1)+1$ は，γ_f^* の下界である．ゆえに，式 5.2 より，

$$\gamma \leq \gamma^* + (\Delta - 1) \cdot \left\lfloor \frac{n-1}{2} \right\rfloor, \tag{5.3}$$

が成り立つ．それゆえ，

$$\begin{aligned} \frac{\gamma}{\gamma^*} &\leqq 1 + \frac{(n-1)(\Delta-1)/2}{\gamma^*} \\ &\leqq 1 + \frac{(n-1)(\Delta-1)/2}{c(n-1)+1} \\ &\leqq 1 + \frac{(n-1)(\Delta-1)/2}{c(n-1)} \\ &= 1 + \frac{\Delta-1}{2c} \end{aligned} \tag{5.4}$$

ただし，$\Delta := \max_{j \neq k \wedge j,k \in I_n}\{w(b_j, a_k)\}$ である．ヒューリスティックアルゴリズムのステップ 4 で，フローの更新に伴うコスト増加量が高々$\Delta - 1$ であることは，以下の事実から導かれる．

- ステップ 3 でコスト 0 の枝を選択すると，コスト増加量は高々Δ である．
- どのサイクルもコストが正の枝が存在するので，その枝をステップ 3 で選択すると，増加量は Δ よりも小さい．
- ステップ 3 での枝の選択のやり方より，フローの変更に伴うコスト増加量は高々$\Delta - 1$ である．

以上の議論から次の様な系を得る．これは，$\Delta = 1$ の時は，式 (5.4) より近似比が 1 になることから導かれる．

系 1. $w(b_k, a_j) \in \{0, 1\}$ ならば，ヒューリスティックアルゴリズムは最適解を返す．

最後に，セットアップタイムに関するある制約の下で，近似比を導く．

補題 7. ステップ 2 [2] で，G' には高々 $\gamma^* - 1$ 個の有向サイクルが存在する．

証明: 背理法でこの補題を証明する．ステップ 2 で，G' に有向サイクルが γ^* 個以上存在すると仮定する．すべてのサイクルは正の重みを持った枝を少なくとも

[2] ヒューリスティックアルゴリズムのステップ 2 を指す．

一つは持つ．それゆえ，一つのサイクルは少なくとも一つ分のタイムスロットに貢献する．それゆえ，最適スケジュールで必要な全体のタイムスロット数は，少なくとも $\gamma^* + 1$ 個である．これは，矛盾である． (証明終)

定理 5. セットアップタイムが $s_{jk} \leq h \cdot L$ (i と j はジョブのインデックス, $i \neq j$, $h \in \mathbb{N} \setminus \{0\}$) という制約の下で，我々のヒューリスティックアルゴリズムは近似比 $h + 1$ の近似アルゴリズムである．

証明: 式 5.3 と上の補題の結果から，

$$\gamma \leq \gamma^* + (\Delta - 1) \cdot (\gamma^* - 1).$$

が成り立つ．また，$0 \leq s_{jk} \leq h \cdot L$ が成り立つので，

$$\Delta \leq h + 1$$

を得る．それゆえ，

$$\gamma \leq \gamma^* + h \cdot (\gamma^* - 1).$$

が成り立つ．それゆえ，

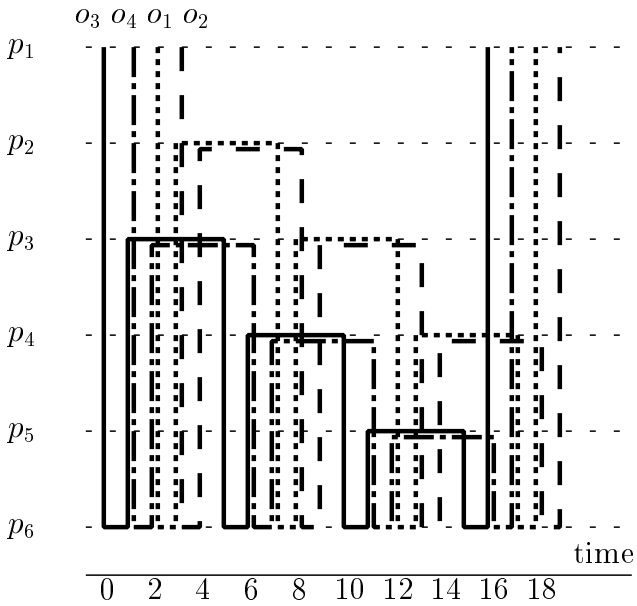
$$\frac{\gamma}{\gamma^*} \leq 1 + h \cdot \left(1 - \frac{1}{\gamma^*}\right) \leq 1 + h.$$

が成り立つ． (証明終)

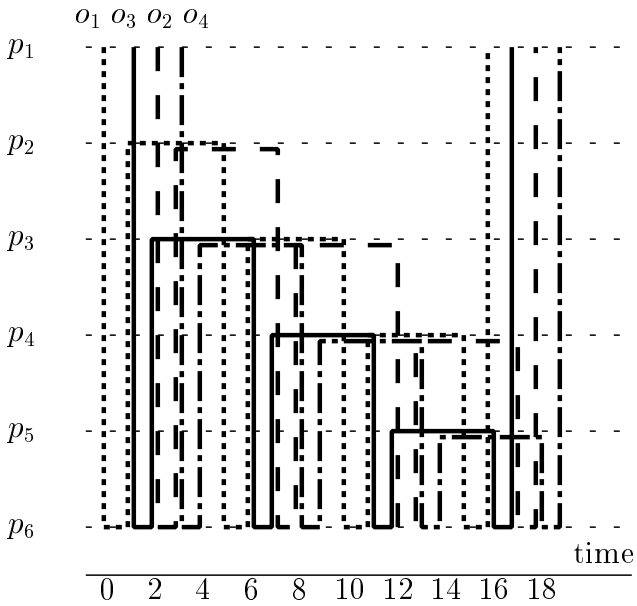
特に，$h = 1$ のとき次の結果を得る．

系 2. すべての順序対 J_j と J_k に対して，セットアップタイムが $s_{jk} \leq L$ という制約の下で，我々のヒューリスティックアルゴリズムは近似比 2 の近似アルゴリズムである．

$h = 1$ の仮定は，近似アルゴリズムを設計する際には適切であろう．このような問題の定式化は [47] で行われた．



(a)



(b)

図 5.7: 搬送計画の例

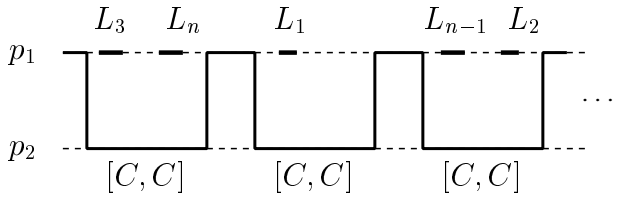


図 5.8: サブルーチンへの入力に対する搬送計画

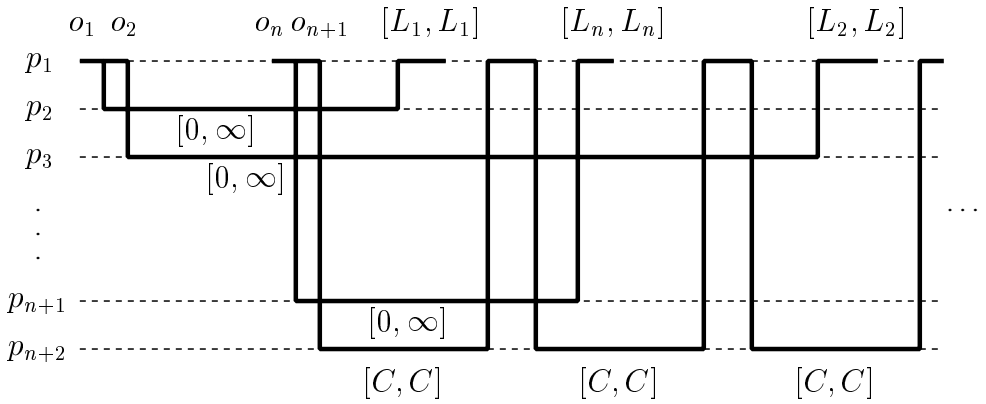


図 5.9: サブルーチンへの入力に対する搬送計画

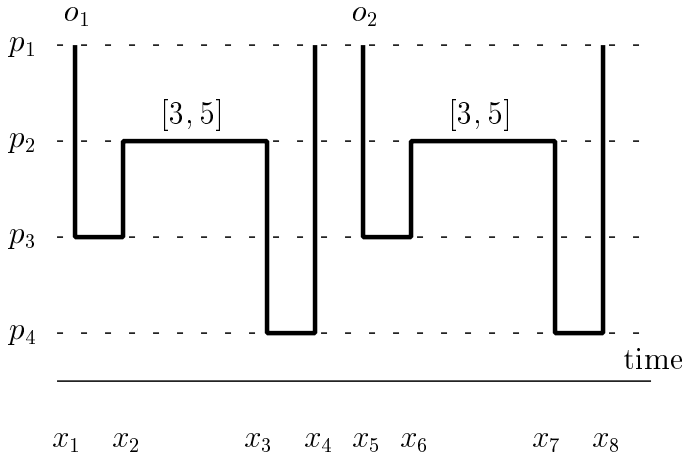


図 5.10: 搬送計画

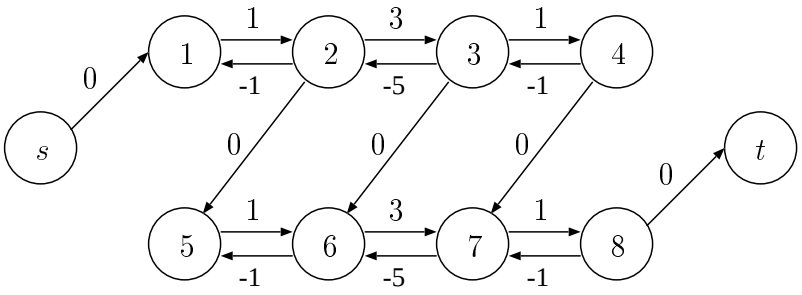


図 5.11: 構成されたネットワーク

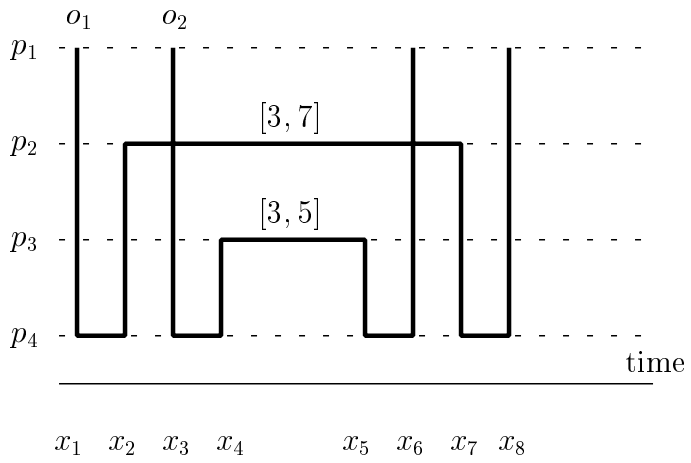


図 5.12: 矛盾のない搬送計画

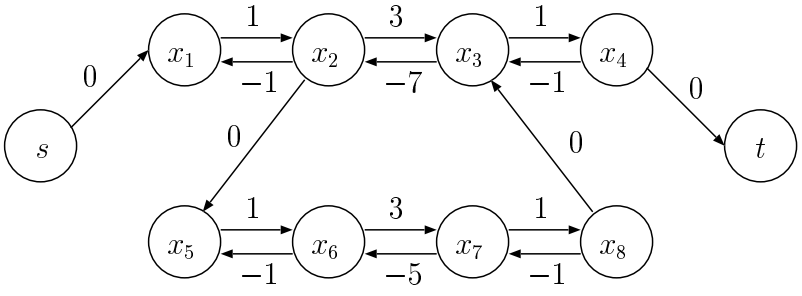


図 5.13: 構成されたネットワーク

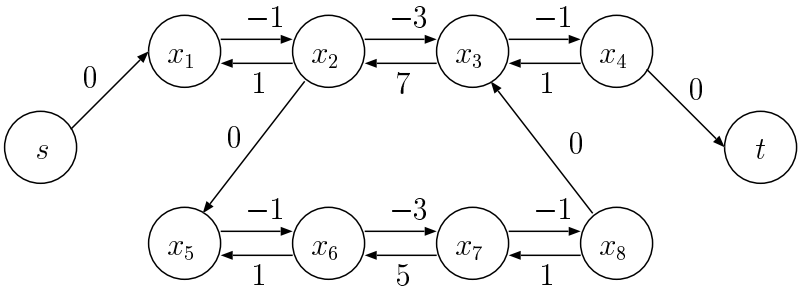


図 5.14: 構成されたネットワーク

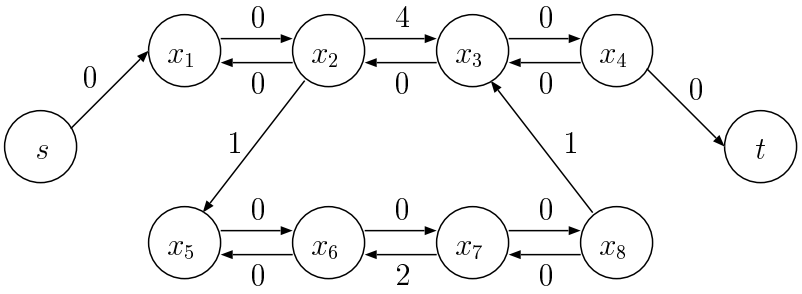


図 5.15: 構成されたネットワーク

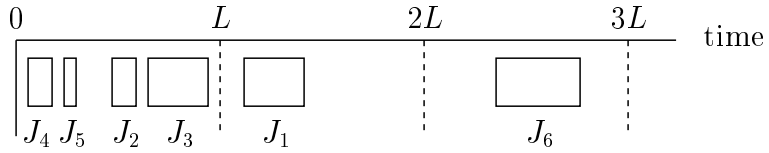


図 5.16: グリーディな手法によるスケジュール

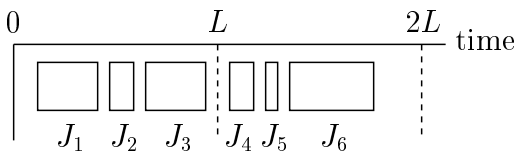


図 5.17: 最適なスケジュール

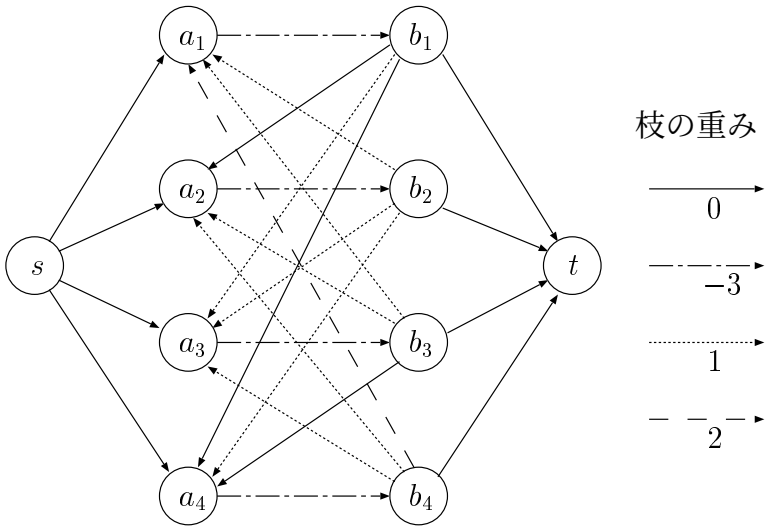


図 5.18: グラフ G の例

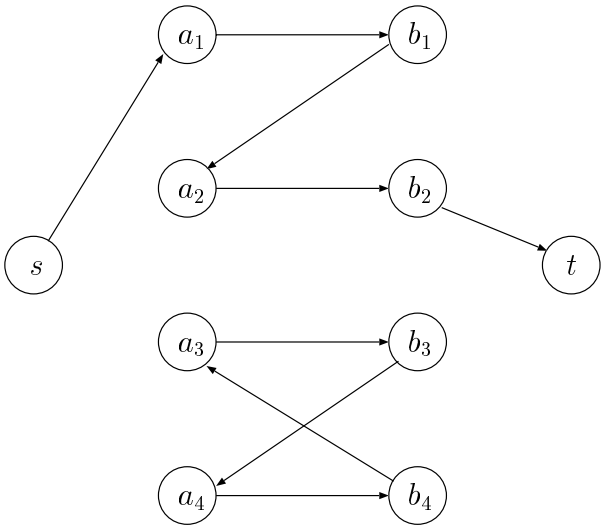


図 5.19: グラフ G' の例

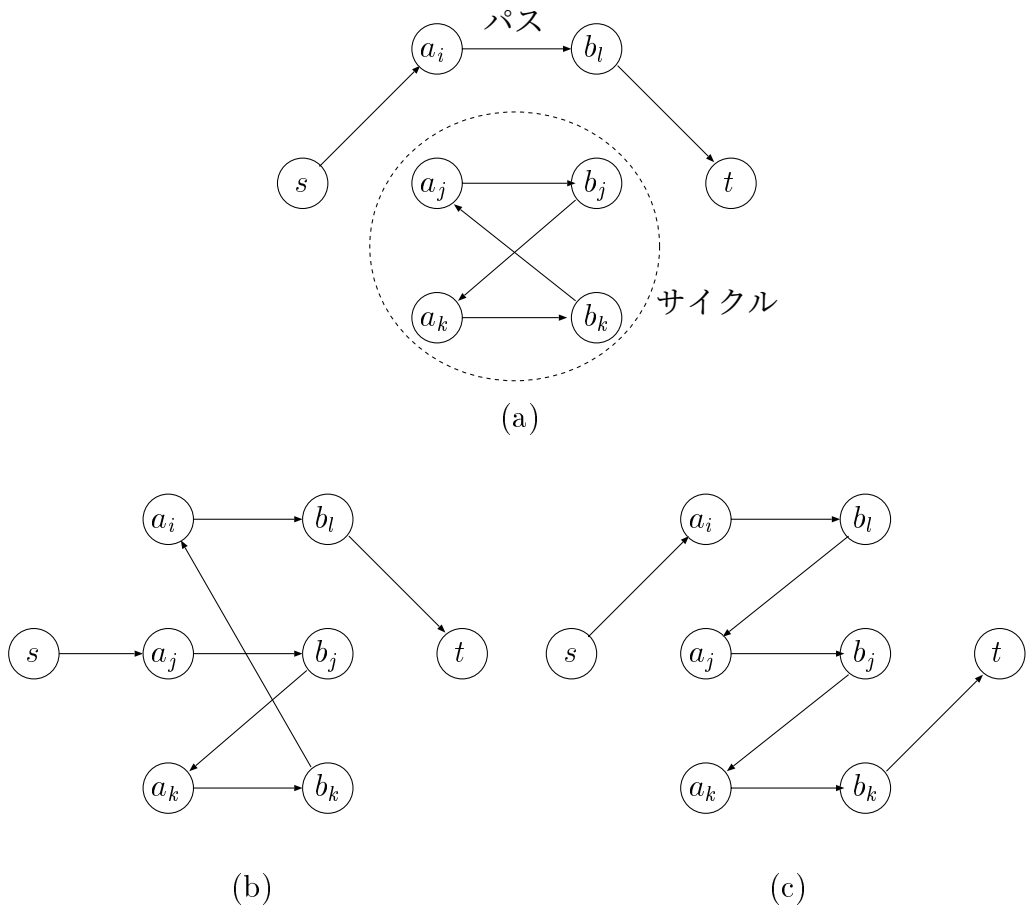


図 5.20: 枝の付け替え

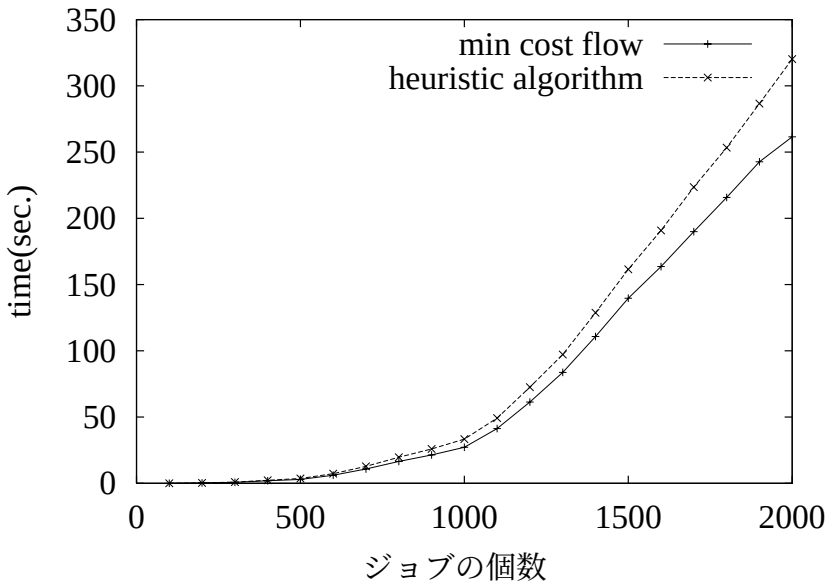


図 5.21: ヒューリスティックアルゴリズムの計算時間

第 6章

むすび

本研究では，液晶や半導体などの製造装置によく見られる直列型の搬送系をモデル化して，搬送系を流れるジョブ同士の衝突確率を解析した．その際，ジョブの処理時間は，実際の状況をよく表している正規分布に従うと仮定した．また，実際の状況をよく表しているわけではないが，処理時間が指数分布に従うと仮定したときも解析して，理論的には大変興味深い結果を得た．正規分布の場合の解析結果は，多重積分を含むものなので，実際に値を計算するときに数値積分などにたよると計算誤差の障害が生じる．一方，指数分布の場合の解析結果は多重積分を含まない．指数分布の性質をうまく利用することで，このように都合の良い結果が得られる．そのため，実際に値を求めるのも容易である．

次に直列型の搬送系上をたくさんのジョブが流れていく過程で，衝突が生じるかどうかをチェックするシミュレーションプログラムを示した．処理時間が正規分布に従う場合は，上で述べたように理論的な解析結果は，多重積分を含んでいるので，具体的な値を求めることがしばしば困難となる．しかし，このシミュレーションプログラムを利用すれば，容易に衝突確率を求めることができる．実際の場面で何らかの目安として使う分には，計算機シミュレーションによる結果で十分であると考える．

また，直列型搬送系で，各機械がバッファを持つときの衝突確率をシミュレーションを利用して求めた．この結果から各機械がバッファを 1 つずつ持つだけで，衝突確率を低く保ったまま，タクトタイムを劇的に小さくすることができることを示した．実際に，処理時間の平均値と同じ値にタクトタイムを設定したとして

も，ほぼ衝突確率は0であった．また，シミュレーション結果から機械の台数を増やしても，衝突確率が増えないという性質を示した．実際の製造装置は数百機械から構成されているとみなすことができるので，この性質はたいへん都合が良いと考える．

最後に，製造の担当者を悩ませている他の問題についても触れた．搬送計画問題は多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，NP 完全であることを証明した．そして，ある制約の下で，多項式時間で解けることを示した．ジャストインタイムスケジューリング問題も製造業に関連している問題である．特に，周期的なタイムスロット付きジャストインタイムスケジューリング問題に対して，高性能なヒューリスティックアルゴリズムを開発した．また，このアルゴリズムは妥当な制約の下で定数近似比を持つ近似アルゴリズムである．

今後の課題としては，直列型搬送系を拡張して，並列型搬送系での衝突確率の解析を考えること，さらに，一般のネットワーク型搬送系での衝突確率の解析を考えることである．第 4.3 章で最も単純な 2 機械並列型搬送系のシミュレーション結果を示したが，並列型の研究はまだ始まったばかりである．また，各機械がバッファを持つときのシミュレーション結果を示したが，理論的に解析できることがないか考えたい．

最終的な目標は，実在の半導体や液晶の製造装置そのものの処理プロセスを計算機上でシミュレーションして，同時に理論的な解析を示すことである．実在の製造装置により近づけるという意味で，今後は第 2.3 章で述べたキャリアの搬送時間，ジョブの物理サイズ，終了センサーだけでなく，バッファへのジョブの出し入れに要する時間なども考慮する必要があると考えている．

謝辞

本研究を行なうに当たり, 終始御指導を賜わった浅野 哲夫教授に深謝致します.

京都大学大学院 工学研究科 加藤 直樹教授には，理論的な面から御教示を賜りました．大日本スクリーン製造株式会社 三塚 郁夫氏には，実用的な見地から様々な御意見を頂きました．深く感謝致します.

本稿の第 5.2 章である副テーマ研究を行うにあたり，有意義な御指導を賜りました，平石 邦彦教授に深く御礼申し上げます．並びに，有益なご助言を賜った Shao Chin Sung 助手（現在，青山学院大学理工学部 経営工学科助教授）に感謝致します.

日頃から様々な貴重な御助言を賜り，また，審査にあたり御教示，御検討頂きました，上原 隆平助教授に深く御礼申し上げます.

京都大学大学院 情報学研究科 伊藤 大雄助教授，北陸先端科学技術大学院大学 情報科学研究科 金子 峰雄教授，石原 哉助教授には，審査にあたり御教示，御検討を賜りましたことを深く感謝致します.

また，日頃から有益な御助言を頂き，多面に渡って励まして頂いた，元木 光雄助手，Arijit Bishnu 助手（現在，Indian Institute of Technology Kharagpur, Kharagpur）に感謝致します.

また，日頃から熱心な議論と多面に渡るご協力を賜った修士課程の三浦 岳くん，並びに情報基礎学講座の学生の皆さん，及び諸先輩方に厚く御礼申し上げます.

最後に在学中の研究活動を暖かく見守ってくれた両親，そして多くの友人たちに心より感謝致します.

参考文献

[1] 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.

[2] 一松信，微分積分学入門第一課，近代科学社，1989.

[3] 大野豊，磯田和男，新版 数値計算ハンドブック，第 12.1.2 章，オーム社，1990.

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

[5] 黒河龍三，初等関数に関する Liouville の研究，数物会誌，1, pp.17–27, pp.146–155, 1927, 3, pp.8–18, pp.285–296, 1929.

[6] 東京大学教養学部統計学教室，統計学入門，東京大学出版会，1991.

[7] 日本経営工学会，生産管理用語辞典, 日本規格協会，2002.

[8] 日本数学会，岩波 数学辞典 第 3 版，岩波書店，1985.

[9] 人見勝人，入門編生産システム工学，第 3 版，共立出版，2005.

[10] 牧野都治，待ち行列の応用，4 章，森北出版，1969.

[11] 森口繁一，宇田川かね久，一松信，岩波 数学公式 I 微分積分・平面曲線，岩波書店，1993.

[12] 森口繁一，宇田川かね久，一松信，岩波 数学公式 II 特殊函数，岩波書店，1992.

[13] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.

[14] A. O. Allen: Probability, Statistics, and Queueing Theory with Computer Science Applications, Second Edition, Section 5.2.1, Academic Press, 1990.

[15] B. Avi-Itzhak, "A Sequence of Service Stations with Arbitrary Input and Regular Service Times", Management Sci., 11, 5, pp.565–571, 1965.

[16] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, NJ, 1982.

[17] D. G. Kendall, "Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain", Ann. of Math. Statistics, vol. 24, pp. 338–354, 1953.

[18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, Sequencing and Scheduling: Algorithms and Complexity, In: S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin (eds.), Handbooks in Operations Research and Management Science, vol.4, Logistics of Production and Inventory, Elsevier, Amsterdam, pp.445–522, 1993.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, 1, pp. 269–271, 1959.

[20] H. Imai, "Notes on the One-Dimensional Compaction Problem of LSI Layouts Viewed from Network Flow Theory and Algorithms", Trans. IECE, Vol. E69, No. 10, pp.1080–1083, 1986.

[21] H. Kise, T. Shioyama, and T. Ibaraki, "Automated Two-machine Flowshop Scheduling: A Solvable Case", IIE Trans., 23, pp.10–16, 1991.

[22] H. T. Papadopoulos, "The throughput of multistation production lines with no intermediate buffers", Operations Research, 43, 4, pp. 712–715, 1995.

[23] `http://groups.yahoo.com/group/lp_solve/files/`

[24] `http://www.jekai.org/entries/aa/00/np/aa00np22.htm`

[25] http://www.misojiro.t.u-tokyo.ac.jp/~tomomi/opt-code.html#LP

[26] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, Scheduling Computer and Manufacturing Processes, Second Edition, Section 3.1, Springer-Verlag, Berlin, 2001.

[27] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", Journal of the ACM, vol. 19, no. 2, pp.248–264, 1972.

[28] J. H. Harris and S. G. Powell, "An Algorithm for Optimal Buffer Placement in Reliable Serial Lines", IIE Trans., 31, pp.287–302, 1999.

[29] J. Liouville, Mémoire sur la classification transcendantes et sur l'impossibilité d'exprimer les racines de certaines équations en fonction finie explicite des coefficients, J. Math. Pures Appl., 2, pp.56–104, 1837, 3, pp.523–546, 1838.

[30] K. Hiraishi, "Scheduling of parallel identical machines with multiple time slots", Proc. 4th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, Jindrichuv Hradec, pp.33–39, Sept. 2001.

[31] K. Hiraishi, E. Levner, and M. Vlach, "Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs", Computers & Operations Research, vol.29, pp.841–848, 2002.

[32] K. -H. Chang and W. -F. Chen, "Admission control policies for two-stage tandem queues with no waiting spaces", Computers & Operations Research, vol.30, pp. 589–601, 2003.

[33] K. Mehlhorn and S. Näher, LEDA: A Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1999.

[34] K. R. Baker and G. D. Scudder, "Sequencing with earliness and tardiness penalties: a review", Operations Research, vol.38, No 1, pp.22–36, 1990.

[35] LEDA HP: http://www.algorithmic-solutions.com/enleda.htm

[36] 日本の LEDA HP: http://www.hulinks.co.jp/software/leda/

[37] MATHEMATICA HP: http://www.wolfram.com/

[38] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms”, Journal of the ACM, vol. 34, pp. 596–615, 1987.

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

[40] M. Schlag, Y. -Z. Liat and C. K. Wong, “An algorithm for optimal two-dimensional compaction of VLSI layouts”, Integration, 1, pp. 179–209, 1983.

[41] M. Sipser, Introduction to the Theory of Computation, PWS publishing company, 1997. (M. Sipser 著，渡辺治ほか訳，計算理論の基礎，共立出版，2000)

[42] M. Y. Hsueh, “Symbolic layout and compaction of integrated circuits”, ERL Memo. UCB/ERL M79/80, Univ. of California, Berkeley, Dec., 1979.

[43] N. G. Hall and C. Sriskandarajah, “A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process”, Operations Research, vol.44, pp.510–525, 1996.

[44] N. Tomizawa, “On some techniques useful for solution of transportation network problems”, Networks, 1, 2, pp. 173–194, 1971.

[45] O. Čepek and S. C. Sung, “A quadratic time algorithm to maximize the number of just-in-time jobs on identical parallel machines”, research report IS-RR-2004-003, School of Information Science, the Japan Advanced Institute of Science and Technology, 2004.

[46] O. Čepek and S. C. Sung, “A Quadratic Time Algorithm to Maximize the Number of Just-in-Time Jobs on Identical Parallel Machines”, Computers & Operations Research, vol.32, pp. 3265–3271, 2005.

[47] O. Čepek and S. C. Sung, "Just in time scheduling with periodic time slots", Proc. 5th Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty, Osaka, Japan, pp.27–29, Sept. 2002.

[48] P. -J. Courtois and G. Scheys, "Minimization of the total loss rate for two finite queues in series", IEEE Transactions on Communications, 39, 11, pp. 1651–1661, 1991.

[49] R. E. Tarjan, Data Structures and Network Algorithms, CBMS Regional Conference Series in Applied Mathematics 44, Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[50] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows — Theory, Algorithms, and Applications, New Jersey: Prentice-Hall, Englewood Cliffs, 1993.

[51] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey", Annals of Discrete Mathematics, vol.5, pp.287–326, 1979.

[52] Stephen Wolfram, The Mathematica Book, 4th ed. , Wolfram Media / Cambridge University Press, 1999. （第 4 版日本語版，東京書籍刊，2000）

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2001.

[54] V. V. Vazirani, Approximation Algorithms, Springer-Verlag, Berlin, 2001. (V. V. ヴァジラーニ 著，浅野孝夫訳，近似アルゴリズム，シュプリンガー・フェアラーク東京，2002）

本研究に関する発表論文

[1] E. Chiba, T. Asano, T. Miura, N. Katoh and I. Mitsuka: “Analysis of Collision Probability in Automatic Transportation Systems”, submitted to IPSJ Journal.

[2] E. Chiba, T. Asano, T. Miura, N. Katoh and I. Mitsuka: “Modeling of Transportation Systems Using Crash Probability”, Proc. 8th Korea-Japan Workshop on Algorithms and Computation, pp.142–149, Seoul, Korea, Aug. 2005.

[3] E. Chiba and K. Hiraishi: “A Heuristic Algorithm for One-Machine Just-In-Time Scheduling Problem with Periodic Time Slots”, IEICE Trans., vol.E88-A, no.5, pp.1192–1199, 2005.

[4] 千葉英史，平石邦彦: “周期的なタイムスロット付きジャストインタイムスケジューリング問題のヒューリスティックアルゴリズム”, IPSJ SIG Technical Reports, 2004-AL-94, pp.1–8, 2004年3月.

[5] T. Asano and E. Chiba: “A New Approach to Transportation Scheduling Problem Based on Network Flow Theory”, Proc. 7th Japan-Korea Workshop on Algorithms and Computation, pp.228–236, Sendai, Japan, Jul. 2003.

[6] 浅野哲夫, 千葉英史: “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”, IEICE Technical Report, COMP2002-49, pp.27–31, 2002年11月.

[7] 千葉英史: “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”, 平成14年度 電気関係学会北陸支部連合大会 講演論文集，pp.220, 2002年9月.