Instituto Superior de Ciências do Trabalho e da Empresa

Uma INTRODUÇÃO ao SCHEDULING

Relatório Interno ISCTE-DCTI-1996
Departamento de Ciências e Tecnologias de Informação do I.S.C.T.E.

por

Joaquim Reis

Departamento de Ciências e Tecnologias da Informação
ISCTE - Instituto Superior de Ciências do Trabalho e da Empresa
Avenida das Forcas Armadas, 1600 Lisboa, Portugal
E-mail: Joaquim.Reis@iscte.pt

Outubro de 1996

1. Introdução

O objectivo deste trabalho é introduzir de forma resumida ao problema do Scheduling, ou Programação das Operações, sem intenção de rever, analisar, ou classificar, detalhada e exaustivamente os métodos de solução propostos pela metodologia clássica para cada tipo de problema de Scheduling ([1]). O texto foi elaborado na fase inicial de recolha de bibliografia básica e estudo do estado do conhecimento do Scheduling no contexto de um trabalho de doutoramento e bàsicamente revê o Scheduling clássico e alguns métodos modernos.

Devido à importância do scheduling em ambientes de produção (fábricas) é grande o peso que lhe foi conferido pela investigação científica. Sendo assim é natural que a linguagem aqui usada se oriente mais para este contexto.

2. O Problema do Scheduling

2.1. História Resumida

É o aparecimento de uma noção de tempo objectiva, mensurável com precisão, que vem permitir a idéia de eficiência e uso económico do tempo, como um recurso. A gestão do tempo e dos recursos passou a ocupar um lugar de primordial importância na organização económica dos processos industriais de produção por alturas da Revolução Industrial (séc. XVIII), por se reconhecer que só uma exploração eficiente dos factores de produção (máquinas, por exemplo) garante o máximo de retorno do capital investido.

Charles Babbage (séc. XVIII) foi um dos primeiros a analisar a economia da produção industrial, na sua obra "On the Economy of Machines and Manufactures" (1832), baseando-se em análises empíricas do processos industriais e fábricas do seu tempo (que chegaram a influenciar os trabalhos de Karl Marx e John Stuart Mill). Taylor no seu livro "Principles of Scientific Management" (1911) lançou as bases científicas da organização racional do trabalho.

As necessidades logísticas da 2ª Guerra Mundial desencadearam uma fase de preocupação com os problemas de planeamento e de tomada de decisão, surgindo a ciência da Investigação Operacional. O método Simplex, base matemática para exprimir muitos problemas de optimização de forma rigorosa, é inventado (1947). Surgem várias abordagens à resolução de problemas de scheduling usando o método Simplex. Uma evolução que a Guerra Mundial propulsionou foi também a dos computadores electrónicos (a partir de 1950). Problemas de gestão de recursos em sistemas operativos, de scheduling de processadores, etc., estimularam igualmente a análise matemática de problemas de scheduling [Froeschl 1993].

2.2. Contexto

A necessidade de técnicas de gestão das operações eficientes, tanto na produção como nos serviços, é, numa economia de mercado global, cada vez mais crítica. De entre as várias disciplinas que constituem este espaço de gestão, o Planeamento e Programação da Produção

[1]Esta revisão, análise ou classificação dos métodos de solução clássicos pode ser feita usando obras como [Blazewicz 1994], [Kan 1976], ou [Lawler 1989], aliás citadas ao longo do texto.

é a área em que mais progressos têm sido obtidos e cuja utilização mais vantagens pode trazer ao gestor [Roldão 1995].

O objectivo geral da gestão das operações da produção numa fábrica, por exemplo, é gerar um comportamento coordenado onde a procura é satisfeita a tempo e com baixos custos de posse. Aliás, a componente Planeamento e Programação da Produção constitui uma variável instrumental que influi em todos os objectivos, ou prioridades competitivas: prazo de entrega, qualidade [2], flexibilidade [3] e custo [Roldão 1995]. Na prática, no entanto, a gestão da produção actual caracteriza-se, em geral, por um frequente incumprimento das datas de entrega e volumes de trabalhos em curso elevados.

O Scheduling, ou Programação, ou Programação das Operações, ou Calendarização [4], tem como objectivo a afectação de recursos no tempo necessários para executar um conjunto de processos [Baker 1974] e é uma área de actividade que se distingue da Gestão de Stocks, ou do Planeamento. Numa actividade produtiva, por exemplo, o scheduling é disjunto da Gestão dos Stocks, pois as decisões de afectação de recursos de produção são o foco principal dos problemas do scheduling e, por outro, lado são tomadas fora do âmbito da gestão de stocks; também o Planeamento da Produção se distingue do Scheduling na medida em que o seu problema essencial é decidir os níveis de recursos de produção necessários, e num horizonte temporal relativamente mais dilatado, decisão essa que é exógena ao Scheduling [Graves 1981].

O Scheduling é uma actividade de gestão de grande importância, não fosse na realidade um ponto central na Economia o uso criterioso de recursos e do tempo. Por isso têm sido desenvolvidas modernamente técnicas/ferramentas automatizadas/computorizadas para um uso melhor optimizado dos recursos e do tempo, não só a nível predictivo (programação da produção a curto prazo, numa base diária, por exemplo), como também a nível reactivo (revisão da programação para reagir a oscilações ou falhas, com o mínimo de disrupções no calendário das operações prèviamente programado).

Apesar destes progressos, a Programação das Operações continua a ser um problema complexo, que resiste a enquadrar-se num modelo genérico, pois as suas características e particularidades variam de caso (fábrica, instalação, empresa) para caso, e num mesmo caso podem mesmo variar com a situação.

2.3. Definição Informal

Apresentam-se algumas definições do Scheduling, de investigadores envolvidos na área.

K. R. Baker

Scheduling tem como objectivo a afectação de recursos no tempo necessários para executar um conjunto de processos [Baker 1974].

---

[2]Entendida como a conformidade dos produtos (ou serviços) face às necessidades expressas pelos clientes.

[3]Entendida como a capacidade de responder a mudanças imprevisíveis (avarias, urgências, variações nos prazos de entrega, no volume de produção, ou no mix de produtos).

[4]Os termos Scheduling, Programação, ou Programação das Operações, e Calendarização, serão aqui usados como sinónimos. O termo Programação, que isolado, tem hoje uma conotação ligada à Programação de Computadores com Linguagens de Programação, apenas terá esta conotação se ela for explicitamente indicada, neste texto. Também os termos schedule, programa, ou calendário, serão usados como sinónimos, referindo-se ao produto da actividade Scheduling, Programação, ou Programação das Operações, ou Calendarização.

Lawler et al

Sequenciação e Scheduling preocupam-se com a afectação óptima de recursos limitados a actividades no tempo [Lawler 1989].

Mark S. Fox

Selecção de planos alternativos e afectação de recursos e tempos a cada actividade, de modo a obedecer a restrições de tempo nas actividades e limitações de capacidade de um conjunto de recursos partilhados [Fox 1994].

Stephen F. Smith

Afectação de recursos a actividades de múltiplos processos independentes ao longo do tempo de modo a optimizar um conjunto de objectivos e preferências [Smith 1994].

Claude Le Pape

Processo de decisão para afectação de recursos ao longo do tempo com vista a realizar uma colecção de tarefas, sujeito a constrangimentos (datas limite, duração e precedência das operações, tempos de movimentação e preparação, disponibilidade e partilha de recursos) e preferências - constrangimentos relaxáveis (relacionadas com datas limite, produtividade, frequência de troca de ferramentas, níveis de stock e estabilidade da fábrica) [Le Pape 1994].

Norman Sadeh

Afectação de recursos (máquinas, ferramentas, operadores humanos) ao longo do tempo a um conjunto de tarefas, atendendo a uma variedade de constrangimentos e objectivos [Sadeh 1994].

Nicola Muscettola

Tem a ver com as operações numa base diária. Dados um conjunto de objectivos instanciam-se os planos e atribui-se a cada acção uma fatia de tempo para uso exclusivo dos recursos necessários. O resultado é uma predição de um curso específico de acção que, ao ser seguido, assegura que se atinjem todos os objectivos respeitando os constrangimentos físicos do sistema [Muscettola 1994].

McDermott & Hendler

Um caso especial importante do planeamento para o qual, dado um conjunto de acções a serem executadas, se tem de produzir a ordem pela qual elas devem ser executadas. Por cada uma das acções requerer um conjunto de recursos de capacidade finita haverá uma preferência de certas ordens em relação a outras [McDermott 1995].

Chase & Aquilano

Um programa é um esquema para a realização de actividades, utilizando recursos ou atribuindo instalações. A finalidade da programação das operações, na instalação funcional, é desagregar o programa director de produção em actividades faseadas por semana, dias ou horas - ou, por outras palavras, especificar, em termos precisos, a carga de trabalho do sistema produtivo planeada a muito curto prazo [Chase 1995].

Burke & Prosser

É o problema de decidir *o que fazer* (operações), *quando* (durações, tempos) e *onde* (recursos), em princípio com o objectivo de maximizar o lucro da empresa. Todos estes elementos estão sujeitos a mudanças (mesmo a forma como o objectivo é conseguido pode variar conforme a conjuntura de mercado). Por isso é melhor construir um sistema de programação baseado em *satisfação*, em vez de optimização, e reactivo, em vez de apenas predictivo, que seja capaz de manter um programa satisfatório num mundo aberto e dinâmico [Burke 1994].

Aspectos a Salientar

Processos, tarefas, operações, actividades ou acções, recursos ou máquinas, tempo, planeamento, curto prazo, objectivos, constrangimentos, preferências estão presentes nas definições apresentadas.

De um modo geral, nos problemas de scheduling assume-se a necessidade de executar um certo número de processos, ou tarefas, cada um dos quais consiste numa dada sequência de operações, a executar pela ordem especificada na sequência, e usa um certo número de máquinas. O processamento de uma operação requer o uso de uma máquina particular durante um certo intervalo de tempo e cada máquina só pode processar uma operação de cada vez. É dada uma função que mede o custo de cada solução possível e pretende-se obter uma solução que cujo custo seja mínimo.

A terminologia usada parece sugerir que o problema ocorre num contexto de produção industrial, mas na realidade ele ocorre em contextos variados. E é fácil verificá-lo se trocarmos os processos e máquinas numa fábrica por pacientes e equipamentos de um hospital, turmas e professores numa escola, navios e docas num estaleiro, programas e computadores, refeições e cozinheiros num restaurante, cidades e caixeiros viajantes.

Um aspecto do problema do Scheduling é o da Sequenciação que tem a ver com a ordenação, ou sequência, das operações em cada máquina. Uma ordenação, ou sequência, que permita que a sequência de operações de cada processo seja processada na ordem correcta é dita sequência possível, ou sequência executável. Pode haver preferências por uma ou outra sequência possível se houver prioridades associadas aos processos. Um programa, schedule, ou calendário [5], é obtido de uma sequência possível especificando os tempos de início (e de fim) exactos para todas as operações [Kan 1976]. Com o scheduling, como já se disse, pretendem-se encontrar programas e, em geral, programas que minimizem determinado custo.

2.4. Definição Formal

2.4.1. Introdução e Notação

O problema de scheduling em geral é posto do seguinte modo (adaptado de [Blazewicz 1994], de [Sadeh 1994], de [Roldão 1995] e de [Kan 1976]):

- Há um conjunto de k *operações*, ou *actividades*, **O** = { O_1, O_2, ..., O_k }.
- Há um conjunto de m *processadores*, *máquinas*, ou *recursos*, **P** = { P_1, P_2, ..., P_m }.

[5] O termo calendário não é, òbviamente, aqui usado no sentido estritamente convencional.

- Podem ainda considerar-se recursos de um conjunto de s *recursos adicionais* **R** = { R_1, R_2, ..., R_s }, se as operações necessitarem de recursos para além dos processadores. Num ambiente de produção estes recursos podem corresponder a trabalhadores, ferramentas, materiais, dispositivos de transporte, etc. (originando um modelo do problema de scheduling mais complexo).

2.4.2. Programação

A *programação*, *scheduling*, ou *calendarização*, corresponde à afectação dos processadores de **P** (e, possivelmente recursos de **R**) às operações de **O**, associando-se a cada uma destas um tempo de início, de modo a completar todas as operações sob os constrangimentos impostos. Em scheduling clássico há dois constrangimentos gerais [Blazewicz 1994, Cap.3], [Lawler 1989] (Part 1.1):

- Cada operação usa, ou é processada por, no máximo, um processador (usando possivelmente quantidades especificadas de recursos adicionais) de cada vez.
- Cada processador pode processar, ou é usado por, no máximo, uma operação de cada vez.

Um outro constrangimento está frequentemente implícto e é o da natureza determinística do problema, quer dizer:

- Toda a informação que define o problema é conhecida de antemão.

2.4.3. Classificação dos Processadores

Os processadores podem classificar-se de acordo com as funções que desempenham, do seguinte modo [Blazewicz 1994]:

Processadores paralelos

Se todos os processadores de **P** desempenham as mesmas funções. Podem ser distinguidos pelas suas velocidades de processamento:

- Processadores paralelos *idênticos* - Todos têm igual velocidade de processamento.
- Processadores paralelos *uniformes* - Diferem nas velocidades de processamento mas, para cada processador essa velocidade não varia, sendo independente da operação.
- Processadores paralelos *não relacionados* - Diferem nas velocidades de processamento, dependendo estas da operação particular.

Processadores dedicados

Os processadores de **P** são especializados na execução de certas operações.

2.4.4. Classificação dos Recursos

Os recursos adicionais podem ser classificados quanto ao seu *tipo* e quanto à sua *categoria*, do seguinte modo [Blazewicz 1994]:

Tipo de recurso

Podem classificar-se de acordo com a funcionalidade preenchida (recursos do mesmo tipo preenchem a mesma funcionalidade).

Categoria de recurso

Podem classificar-se segundo dois pontos de vista:

Quanto aos **constrangimentos de recurso**, um recurso pode ser:

- **Renovável** - Aquele em que apenas a sua disponibilidade temporária, ou uso total, em cada momento, é restrito, isto é, ao ser usado por uma operação, o recurso ficará novamente disponível após a operação o libertar.
- **Não renovável** - Aquele em que apenas a sua disponibilidade integral, ou consumo total, até um certo momento, é restrito, isto é, uma vez usado por uma operação o recurso não mais pode ser usado.
- **Duplamente constrangido** - Aquele em tanto o uso total, em cada momento, como o consumo total, até um certo momento, são restritos.

Quanto à **divisibilidade do recurso**, pode ser:

- **Discreto** - Se é discretamente divisível, isto é, pode ser afectado a operações numa das quantidades discretas de um conjunto finito de afectações possíveis (de que a afectação de uma única unidade de recurso por operação é um caso particular).
- **Contínuo** - Se é continuamente divisível, isto é, pode ser afectado numa quantidade arbitrária, inferior ou igual a uma certa quantidade máxima.

2.4.5. Caso dos Processadores Dedicados

Quando se trata do caso de processadores dedicados, o conjunto de operações é particionado em subconjuntos denominados *processos* ou *jobs*. O problema é então, assim posto:

- Há um conjunto de n *processos* $\mathbf{J} = \{ J_1, J_2, ..., J_n \}$.
- Cada processo J_j consiste num conjunto de k_j *operações* $\mathbf{O}^j = \{ O^j_1, O^j_2, ..., O^j_{kj} \}$.
- As operações usam, ou são processadas por, *processadores* de um conjunto de m processadores $\mathbf{P} = \{ P_1, P_2, ..., P_m \}$.

Os seguintes parâmetros são também usados para caracterizar os processos e as suas operações:

- Cada processo tem uma *data de início*, ou *tempo de início*, r_j (correspondente à chegada da encomenda, ou à chegada de matérias primas necessárias para produzir a encomenda), uma *data limite*, ou *tempo limite*, d_j (de entrega), e uma *data terminal*, t_j (correspondente, por exemplo à data em que o cliente recursará a encomenda), sendo $r_j \le d_j \le t_j$.
- A cada processo pode associar-se um *peso*, ou *importância*, w_j (significando, por exemplo, um custo por unidade de tempo).
- Poderá ser especificada uma *ordem*, ou *constrangimento temporal de precedência*, entre operações de cada processo (exemplo: $O^j_a < O^j_b$ indica que O^j_b só poderá iniciar-se após O^j_a terminar).
- Existem de *acumuladores*, *stocks intermédios*, ou *buffers*, entre os processadores em cada processo que, em geral, se assume terem capacidade ilimitada (e, assim sendo, um processo após terminado num processador pode esperar até que o processamento no seguinte se inicie). Pelo contrário, assume-se a propriedade (dos processos) de *não-espera*,

ou *no-wait*, se têm capacidade zero (o processos não podem esperar entre dois processadores consecutivos).

- Para cada operação O^j_i há uma *duração*, ou *tempo de processamento*, fixo, p^j_i.
- As operações podem, ou não, ser interrompidas e retomadas mais tarde (no mesmo processador ou noutro). No primeiro caso a programação e o programa resultante são ditos *preemptivos*, no outro caso *não preemptivos*.
- Cada operação necessita de um único processador de **P** (para o qual poderão, ou não, existir várias alternativas) e (possivelmente) de recursos adicionais de **R**.
- A cada operação associa-se O^j_i um tempo de início s^j_i, cujo valor numérico será estabelecido ao construir o programa. O domínio de valores possíveis para cada s^j_i pode ser definido através de uma data de início mais cedo r^j_i e uma data de início mais tarde rt^j_i. Estas podem ser derivadas da data de início r_j, com $r^j_1 = r_j$, e preferencialmente, da data limite d_j, com $rt^j_{kj}+p^j_{kj} = d_j$ (a não ser excedida sob pena de o processo J_j sofrer atraso) ou, alternativamente e em caso extremo, da data terminal t_j, com $rt^j_{kj}+p^j_{kj} = t_j$. Também se pode definir de um modo semelhante uma *data limite*, ou *tempo limite*, d^j_i para cada operação O^j_i.

Os parâmetros s^j_i terão valor apenas quando o programa tiver sido construído.

Alguns parâmetros, úteis para traduzir critérios de scheduling, são derivados dos anteriores. São, para cada processo J_j:

- *Tempo de fim*, ou *completion time*, C_j
- *Tempo de espera*, ou *waiting time*, $W_j = C_j - (r_j + \sum_{j=1,k_j} p^j_i)$
- *Tempo de fluxo*, ou *flow time*, $F_j = C_j - r_j$ (soma dos tempos de espera e processamento)
- *Atraso relativo*, ou *lateness*, $L_j = C_j - d_j$
- *Atraso absoluto*, ou *tardiness*, $Dj = \max\{ C_j - d_j, 0 \}$

Pressupondo que o conjunto de operações está particionado em subconjuntos de acordo com os processadores respectivos - deste modo O^q_i , para i=1 a k_q, serão todas as operações afectadas ao processador P_q - o seguinte parâmetro pode também ser definido para cada processador P_q:

- *Tempo de paragem*, ou *tempo de recurso parado*, ou *idle time*, $I_q = Cmax - \sum_{i=1,k_q} p^q_i$

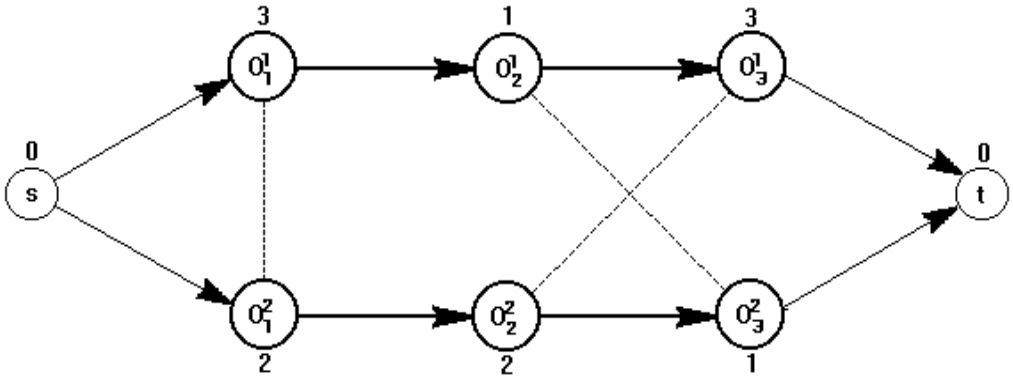
2.4.6. Programa

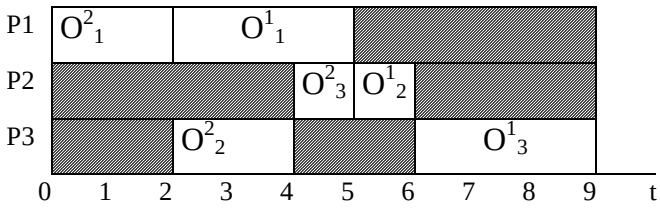
Um *programa*, *schedule*, ou *calendário*, é uma afectação de processadores do conjunto **P** (e, possivelmente de recursos adicionais do conjunto **R**) às operações dos conjuntos $\mathbf{O}^j$ (ou do conjunto **O**), nos tempos s^j_i para cada operação, tais que [Blazewicz 1994]:

- Em cada momento cada processador está afectado no máximo a uma operação, e cada operação é processada por, ou usa, no máximo um processador.
- Cada operação O^j_i é processada só após estar disponível para processamento, isto é, tem de ser $s^j_i \in [r^j_i, \infty)$.
- Todas as operações terminam.

- Se há constrangimentos de precedência $O^j_a < O^j_b$, o processamento de O^j_b não pode iniciar-se antes de O^j_a terminar, isto é, $s^j_a + p^j_a \leq s^j_b$.
- No caso de programação não preemptiva nenhuma operação é interrompida ou, no caso de programação preemptiva o número de interrupções de cada operação é finito.
- São satisfeitos os constrangimentos de recurso, se os houver.

Quando há constrangimentos de precedência, o conjunto de operações ordenado pela relação de precedência pode ser representado pelo *grafo de precedências*, um grafo direccionado cujos nós correspondem às operações e cujos arcos correspondem às relações de precedência, sem arcos transitivos. Para representar programas é costume usarem-se os chamados *mapas de Gantt*.





Na figura acima representa-se um exemplo de grafo de precedência para um problema com 2 processos com 3 operações cada, com 3 processadores dedicados. Os nós s e t representam operações fictícias que delimitam o horizonte temporal de programação. Junto de cada nó é indicada a duração da operação respectiva. Mostra-se também o mapa de Gantt de um programa que é a solução óptima para o problema.

Para além dos arcos de precedência, são adicionalmente incluídos, no grafo representado, arcos não direccionados (a tracejado) que ligam operações que requerem o mesmo processador. Por isto o grafo é denominado *grafo disjuntivo*. Arcos não direccionados especificam disjunção das operações que eles ligam relativamente a um recurso, isto é, que não mais de uma dessas operações poderá usar o recurso ao mesmo tempo.

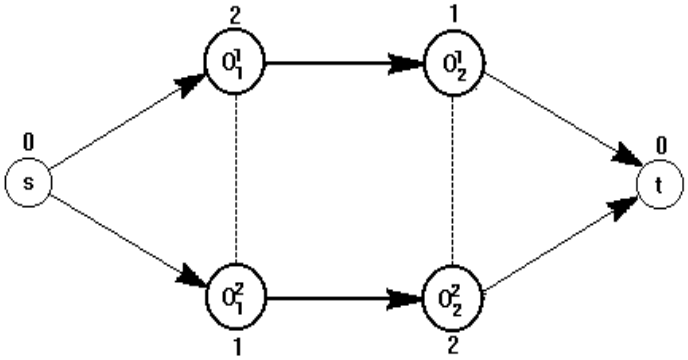
Um *programa sem-atraso* (ver exemplo na figura abaixo) é aquele em que cada processador não tem tempos de paragem, à excepção de antes de iniciar, ou após terminar, o processamento das operações a ele afectadas.

Um *programa semi-activo* (ver exemplo na figura abaixo) é um programa tal que não é possível reduzir o tempo de início de qualquer operação sem alterar a ordem das operações

em alguma máquina. É um programa (ou um dos programas) com tempos de início mais cedo para cada operação (calculáveis através do grafo disjuntivo). Para n processos e m máquinas há $n!^m$ programas semi-activos.

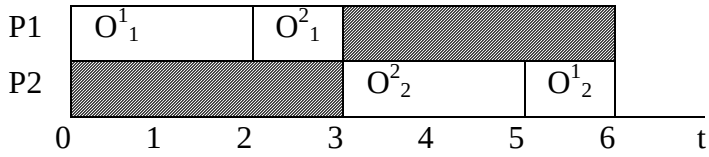
Um *programa activo* (ver exemplo na figura abaixo) é um programa semi-activo no qual não é possível reduzir o tempo de início de qualquer operação sem aumentar o de, pelo menos, uma outra operação. O conjunto de programas activos para um dado problema (que é um subconjunto dos conjunto de programas semi-activos) é o que contém programas óptimos (se existirem).

Grafo disjuntivo:

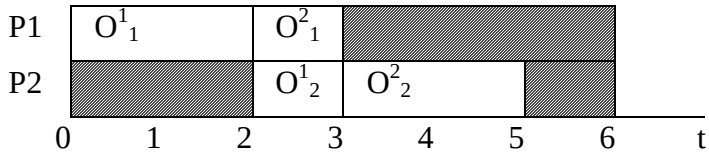


Alguns programas possíveis (em mapa de Gantt), todos sem-atraso:

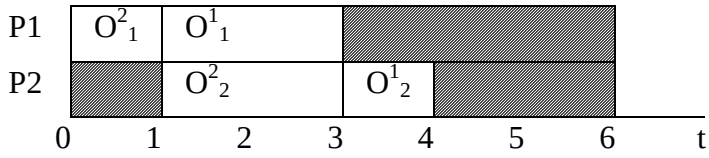
1- Programa semi-activo:



2- Programa activo:



3- Programa óptimo (para o critério C_{max}):



2.4.7. Objectivos de Programação

Há um certo número de *objectivos de programação* cuja satisfação pode ser medida por critérios de avaliação de programas. Um objectivo traduz-se, de um modo muito geral, em encontrar o programa óptimo no que respeita a um critério (ver secção a seguir sobre os critérios de scheduling). Este critério é designado por critério de optimalidade. O problema é assim traduzido num problema de optimização [6].

Por exemplo, no que respeita a uma unidade produtiva tomam-se, em geral, como objectivos do scheduling, os que traduzam as seguintes necessidades [Roldão 1995]:

- Cumprir (ou minimizar o atraso relativamente a) prazos de entrega (as datas limite).
- Reduzir os trabalhos em curso.
- Melhorar a eficiência de utilização dos recursos.
- Reduzir tempos de fluxo.

Na secção respeitante à categorização dos problemas de scheduling serão vistos mais em detalhe os critérios mais usados.

2.4.8. Problema de Programação, Algoritmo de Programação

Um *problema de programação*, *problema de scheduling*, ou *problema de calendarização*, é um conjunto de parâmetros como os definidos anteriormente nesta secção [7], alguns deles sem valores numéricos atribuídos, juntamente com um critério de optimalidade. Uma instância do problema é obtida quando se especificam valores particulares para todos os parâmetros do problema [8][Blazewicz 1994].

Um *algoritmo de programação*, ou *algoritmo de scheduling*, é um algoritmo que constrói um programa para um dado problema, isto é, determina os valores para os tempos de início s de todas as operações. Em geral os problemas são postos como problemas de optimização, usando-se, para encontrar a solução, um *método exacto* que faz uso de um *algoritmo de optimização*. Estes algoritmos são assim designados porque encontram a solução óptima para o problema. Infelizmente, devido à inerente complexidade de muitos problemas de optimização - muitos são NP-difíceis - nem sempre é possível esses algoritmos encontrarem a solução óptima em tempo útil. Podem então usar-se *métodos aproximados*, que encontram soluções aproximadas do problema, isto é, soluções que são subóptimas em relação ao critério usado.

2.5. Uma Categorização dos Problemas de Scheduling

2.5.1. Introdução

Dada a grande variedade de problemas de scheduling é normalmente usada uma notação sistemática que serve para indicar o tipo de problema e serve como base de classificação. Por

[6] No entanto, é também possível formular o problema numa versão de decisão. Por exemplo, o problema de, dada uma série de datas terminais, determinar (se existe) um programa sem atrasos, é um problema de decisão.

7Excluídas as datas de início das operações.

[8]Excluídas as datas de início das operações.

exemplo, a notação apresentada em [Blazewicz 1994], usa três campos, α, β e γ, para especificar um determinado problema de scheduling em que o campo α descreve o tipo e número de processadores, o campo β descreve as características das operações e dos recursos adicionais (se existirem) e o campo γ indica o critério de scheduling utilizado no problema.

O objectivo deste texto não é rever, analisar e classificar detalhada e exaustivamente os métodos de solução propostos pela metodologia clássica para cada tipo de problema de scheduling (o que, dada a variedade possível o tornaria pesado). Por isso se opta por uma forma geral e sintética de categorização que possa abranger características gerais do scheduling, tanto na teoria como na prática, como a apresentada em [Graves 1981]. Este autor identifica dimensões segundo as quais, tipos de problema de programação da produção diversos podem categorizar-se. Três dessas dimensões, que nos parecem ser importantes, são:

- **Complexidade do processamento** - Tem bàsicamente a ver com o número de passos de processamento associados a cada tarefa de produção.
- **Critérios de scheduling** - Tem a ver com as medidas pelas quais serão avaliados os programas gerados.
- **Natureza estática/determinística ou dinâmica/não determinística do problema** - Tem a ver com a disponibilidade ao longo do tempo dos valores precisos dos parâmetros do problema e com as suas possíveis oscilações ou variações.

2.5.2. Complexidade do Processamento

Os ambientes de produção podem ser classificados quanto ao grau de continuidade do processo usado na instalação fabril, originando problemas de programação que diferem. Com os processadores em **P** dedicados, isto é, especializados na execução de certas operações há, bàsicamente, dois ambientes possíveis [9]:

- **Ambiente contínuo**, ou **flow-shop** - O número de operações por processo (k_j) é igual ao número de processadores m e as operações usam os processadores sempre pela mesma ordem. Quer dizer, para cada processo J_j, é executada primeiro a operação O^j_1 usando o processador P_1, depois a operação O^j_2 usando o processador P_2, etc., etc., por último a operação O^j_m usando o processador P_m. É o tipo de produção característico das linhas de montagem e indústrias de processo (refinarias, siderurgias, etc.). Uma variante designada por **ambiente aberto**, ou **open-shop**, é idêntica mas com a ordem de processamento das operações não especificada.
- **Ambiente intermitente**, ou **job-shop** - O número e a ordem de execução das operações pode variar de processo para processo (embora seja fixo à partida). É o tipo de produção característico da produção por lotes e da produção unitária.

Em termos de complexidade de processamento (variações em número, ordem e alternativas à ordem das operações dos processos) o problema do ambiente de produção intermitente, ou job-shop, corresponde ao problema de scheduling mais genérico (o problema do ambiente contínuo pode ser considerado um caso particular deste) e mais complexo. É aquele que mais desafia os métodos clássicos de solução, por conduzir a modelos de optimização combinatórial enormes e difíceis [Graves 1981], [Shapiro 1993], [Blazewicz 1994], Cap 6.3 e 8.3],

[9] Outros casos possíveis além dos descritos são os de ambientes mistos e ambientes flexíveis (com alto nível de automatização e integração entre máquinas, transporte e armazenamento).

mais se presta à utilização de métodos de solução heurísticos. Os seguintes factores podem, no entanto, aumentar ainda mais a complexidade deste tipo de problemas de scheduling:

- Os processos podem ter sequências de operações alternativas (há processos sem ordem entre operações fixa, à partida).
- A mesma operação tem processadores alternativos (por exemplo, uma operação pode ser executada numa de um grupo de máquinas com características semelhantes, podendo até existir uma preferência por uma delas).
- Cada operação necessita de recursos adicionais (por exemplo, para além de uma máquina, podem ter de se considerar, por serem limitados, recursos adicionais como mão-de-obra, ferramentas, matérias primas).
- Há processadores/recursos que podem ser usados por várias operações simultâneamente.
- São tidos em conta os tempos de preparação de um processador/recurso antes da execução de uma operação com ele (que poderão variar conforme a sequência de operações no processador/recurso). Também os tempos de movimentação entre operações poderão considerar-se.
- Uma tarefa pode ser interrompida (por exemplo, devido a uma avaria, ou por outra razão) e mais tarde concluída (programação preemptiva).
- O conjunto de processos não é fixo, ou o conjunto de máquinas pode variar.
- Há que considerar o aspecto dinâmico/não determinístico do problema, isto é, ter em conta factores desconhecidos de antemão e incerteza no problema, tais como: tempo de chegada dos processos para execução (por exemplo, numa fábrica que não funcione apenas para manter stock, a chegada de novas encomendas depende do exterior, e condiciona a fixação de datas de entrega), indisponibilidade dos processadores/recursos (possibilidade de avarias de máquinas, falta de mão-de-obra devido a baixas do pessoal, etc.), cancelamento de processos (por exemplo, devido ao cliente ter cancelado uma encomenda), flutuações nas durações das operações.

2.5.3. Critérios de Scheduling

Um critério é uma medida para avaliar programas. A um critério corresponde uma função de custo que, para cada programa particular, terá um valor. Os valores da função de custo permitem comparar programas diferentes: o programa melhor será o que tem um valor melhor da função de custo. A procura do melhor programa (o programa óptimo) para uma dada instância de problema de programação traduzir-se-á na procura do programa cujos parâmetros originam o menor valor da função de custo.

Segundo [Graves 1981], na prática os critérios para avaliação de um programa podem baseiar-se numa mistura de medidas de custo e de eficiência. No entanto, a maior parte da literatura teórica de scheduling da produção dedicou-se a problemas com um único critério.

Em [Kan 1976], por exemplo, encontramos critérios classificados nas seguintes classes gerais de critérios:

Critérios Baseados nos Tempos de Fim

Baseiam-se nas datas de terminus dos processos, a minimizar. Exemplos (de [Kan 1976]):

- **Comprimento do programa**, ou **makespan**, C_{max} - Tempo de fim programado para o último processo a terminar (contado a partir do tempo zero, ou início do horizonte temporal de programação):

  $C_{max}= \max\{C_j\}$

  Segundo [Blazewicz 1994] a minimzação de C_{max} (que corresponde à realização de um conjunto de operações no menor tempo possível) é importante do ponto de vista do proprietário dos processadores, visto levar à maximização da utilização dos processadores (dentro do comprimento do programa, C_{max}) e à minimização do tempo intra-processo máximo do conjunto de operações programadas.

  Soma dos comprimentos, C_{soma} -

  $C_{soma}= \Sigma_{j=1,n} C_j$

  Soma ponderada dos comprimentos, C_w -

  $C_w= \Sigma_{j=1,n} w_j C_j$

  Comprimento médio, C_{med} - Média dos comprimentos. Igual ao anterior mas em que w_j = 1/n, para todo o j.

  $C_{med}= (\Sigma_{j=1,n} C_j)/n$

- **Soma dos tempos de fluxo**, F_{soma} -

  $F_{soma}= \Sigma_{j=1,n} F_j$

  Soma ponderada dos tempos de fluxo, F_w -

  $F_w= \Sigma_{j=1,n} w_j F_j$

  Tempo de fluxo médio, F_{med} - Média dos tempos de fluxo. Igual ao anterior mas em que w_j = 1/n, para todo o j.

  $F_{med}= (\Sigma_{j=1,n} F_j)/n$

  Segundo [Blazewicz 1994] a minimzação de F_{med} corresponde à minimização do tempo médio de resposta e do tempo intra-processo médio do conjunto de operações programadas sendo, por isso, importante do ponto de vista do utilizador/cliente.

- **Soma dos tempos de espera**, W_{soma} -

  $W_{soma}= \Sigma_{j=1,n} W_j$

  Soma ponderada dos tempos de espera, W_w -

  $W_w= \Sigma_{j=1,n} w_j W_j$

  Tempo de espera médio, W_{med} - Média dos tempos de espera. Igual ao anterior mas em que w_j = 1/n, para todo o j.

  $W_{med}= (\Sigma_{j=1,n} W_j)/n$

Critérios Baseados nas Datas Limite

Baseiam-se nas datas limite e são medidas de atrasos, a minimizar, sendo de grande importância em sistemas de produção, em especial os que produzem para satisfazer encomendas específicas de clientes. Exemplos (de [Blazewicz 1994] e [Kan 1976]):

- **Atraso relativo máximo**, L_{max} - O maior atraso relativo:

  $L_{max}= \max\{L_j\}$

  Soma dos atrasos relativos, L_{soma} -

  $L_{soma}= \Sigma_{j=1,n} L_j$

  Soma ponderada dos atrasos relativos, L_w -

  $L_w= \Sigma_{j=1,n} w_j L_j$

  Atraso relativo médio, L_{med} - Média dos atrasos relativos. Igual ao anterior mas em que $w_j = 1/n$, para todo o j.

  $L_{med}= (\Sigma_{j=1,n} L_j)/n$

- **Atraso absoluto máximo**, T_{max} - O maior atraso absoluto:

  $T_{max}= \max\{T_j\}$

  Soma dos atrasos absolutos, T_{soma} -

  $T_{soma}= \Sigma_{j=1,n} T_j$

  Soma ponderada dos atrasos absolutos, T_w -

  $T_w= \Sigma_{j=1,n} w_j T_j$

  Atraso absoluto médio, T_{med} - Média dos atrasos absolutos. Igual ao anterior mas em que $w_j = 1/n$, para todo o j.

  $T_{med}= (\Sigma_{j=1,n} T_j)/n$

Critérios Baseados em Custo e Utilização

Têm em conta custos de posse (existências) e a utilização dos recursos. Exemplos [Kan 1976]:

- **Número médio de processos em curso**, N_p, **em espera**, N_e, ou **acabados**, N_f - Valor médio do número de processos que respectivamente, têm uma operação em curso, estão em espera, ou terminaram, tomado ao longo do intervalo de tempo correspondente ao comprimento do programa.
- **Média do trabalho em progresso**, A - Valor médio da soma das durações de cada operação em curso, tomado ao longo do intervalo de tempo correspondente ao comprimento do programa.
- **Soma dos tempos de paragem**, ou **tempo de paragem total**, I_{soma} - Soma dos tempos de paragem de todos os recursos dentro do intervalo de tempo correspondente ao comprimento do programa

$I_{soma} = \sum_{q=1,m} I_q$

Soma ponderada dos tempos de paragem, I_v - Referindo-se aqui o peso v_q ao processador P_q (pode significar um custo por unidade de tempo de paragem):

$I_v = \sum_{q=1,m} v_q I_q$

- **Utilização média**, U_{med} -

  $U_{med} = (\sum_{j=1,n} \sum_{i=1,k_j} p^j_i)/(m * C_{max})$

Segundo [Kan 1976], a minimização de N_e, ou de $N_p + N_e$, reflectirá a necessidade de minimizar os stocks intermédios (de trabalhos em curso), enquanto que a minimização de N_f reflectirá a necessidade de minimizar stocks de produtos acabados. Já a maximização de N_p, A, ou a minimzação de I_{soma}, I_v ou a maximização de U_{med} reflectirá a necessidade de utilização eficiente das máquinas disponíveis.

Segundo [Kan 1976] demonstra, existem as equivalências [10] entre critérios mostradas no seguinte quadro:

Equivalências entre critérios de programação (critérios equivalentes na mesma coluna):

C_{max}	(min)	C_{soma}	(min)	C_w	(min)	$N_p + N_e$	(min)	N_e	(min)
N_p	(max)	F_{soma}	(min)	F_w	(min)	N_f	(min)	W_{med}/C_{max}	
A	(max)	W_{soma}	(min)	W_w	(min)	C_{med}/C_{max}			
I_{soma}	(min)	L_{soma}	(min)	L_w	(min)				
I_v	(min)	C_{med}	(min)						
U_{med}	(max)	F_{med}	(min)						
		W_{med}	(min)						
		L_{med}	(min)						

E ainda:

- Um programa óptimo relativamente a L_{max} (a minimizar) é também óptimo relativamente a T_{max} (a minimizar). Repare-se que L_{max} e T_{max} não são equivalentes: um programa com $T_{max} = 0$ pode ser subóptimo relativamente a L_{max}.

Ainda segundo o mesmo autor, os critérios C_{max}, C_{soma}, C_w, L_{max}, T_{soma} e T_w podem representar todos os outros (dadas as equivalências). Daqui que a literatura sobre scheduling frequentemente refira (por vezes unicamente) alguns destes.

2.5.4. Natureza Estática/Determinística ou Dinâmica/não Determinística do Problema

Um aspecto do scheduling que pode ser importante na prática é o da natureza dinâmica e não determinística do problema. Se os valores dos parâmetros que definem o problema são todos

[10] No sentido de que um programa óptimo para um critério será também um programa óptimo para um critério equivalente.

conhecidos de antemão (natureza estática) e não sofrem variações ao longo do tempo (natureza determinística), o problema é um de *scheduling estático/determinístico*.

Em certos casos pode não fazer sentido assumir isto, pois, pelo contrário, há informação que define o problema não conhecida à partida, ou incerta. Por exemplo, podem aparecer processos novos, ou ocorrer o cancelamento de outros, o valor preciso da duração de uma operação pode ter flutuações, pode haver interrupções devidas a avarias, ou falta de recursos adicionais necessários.

Uma abordagem utilizada para o *scheduling dinâmico*, por exemplo, quando há eventos como chegada de processos novos que ocorrem em tempos desconhecidos, é a de resolver um problema de scheduling estático em cada tempo de ocorrência de tal evento e implementar a solução num horizonte de programação que se repete.

No caso da incerteza, o problema pode ser considerado de *scheduling não determinístico* (ou *estocástico*) podendo ser tratado de um ponto de vista probabilístico. Num modelo de problema deste tipo há parâmetros (ditos estocásticos) que são vistos como variáveis aleatórias independentes, com certas distribuições de probabilidade, e cujo valor só será conhecido depois da decisão de programação [Lawler 1989] (Part 16.1).

De um modelo de problema de scheduling determinístico podem derivar-se vários modelos de problemas de scheduling estocásticos, já que há várias hipóteses de escolha dos parâmetros aleatórios, das suas distribuições de probabilidade e do tipo de tratamento a dar ao problema. Muitos casos de problemas de scheduling determinístico em que, como método de solução, é usado um algoritmo heurístico, que usa uma determinada regra heurística (que não garante uma solução óptima), têm uma reformulação estocástica em que se pode garantir encontrar a solução óptima para o modelo de problema estocástico. Em geral a regra, ou regras, óptimas, a aplicar, em cada ponto de decisão podem mudar, requerendo-se informação sobre o curso até ao momento presente. Os suportes teóricos, são nestes casos, a teoria de decisão semi-Markoviana e a optimização dinâmica estocástica [Lawler 1989] (Part 16.1).

3. Métodos de Solução

3.1. Métodos de Solução Exactos

3.1.1. Introdução

Um método para encontrar a solução óptima para um problema de scheduling é o de usar um algoritmo que procura determinar o programa óptimo de uma forma exacta. Descrevem-se a seguir métodos deste tipo.

3.1.2. Enumeração Completa

Uma forma de encontrar o programa óptimo, que parece intuitivamente natural, pode ser pela *enumeração completa* dos programas possíveis para um determinado caso de problema de scheduling, isto é, construindo e avaliando cada programa possível segundo o critério especificado, e concluir por fim devolvendo o melhor dos programas (ver o algoritmo devido a Giffler & Thompson em [Kan 1976] Cap.3.1 que permite gerar uma vez cada programa activo possível). No entanto, esta forma de abordar o problema não permitirá encontrar, para todos os casos, a solução óptima em tempo útil, pois em geral, o número de programas pode ser muito elevado (mesmo para um problema pequeno).

Por exemplo, segundo [Kan 1976] para um problema geral com n processos e m máquinas haverá não mais de $n!^m$ programas (semi-activos) possíveis diferentes. Mesmo para valores relativamente pequenos de n e m aquele número limite é elevado: para n = 5 e m = 5, por exemplo, o valor de $n!^m$ é 24.883.200.000. Um sistema que possa gerar e avaliar 100.000 programas por segundo precisaria de quase 3 dias para o fazer para todos e poder determinar qual o programa óptimo. Também se pode verificar que à medida que se aumenta a dimensão do problema (mais processos, mais máquinas) o número de programas possível aumenta exponencialmente. Deste modo, a enumeração completa parece ser um método cuja aplicabilidade fica restrita a casos especiais de problemas de scheduling de dimensão reduzida, e em geral, casos sem interesse prático.

3.1.3. Programação Matemática

Um método que merece destaque, pela forma matemática e elegante com que permite formular o problema é o da *programação matemática*.

Um modelo possível usando este método ([Bowman 1959], descrito em [Kan 1976]) serve-se de variáveis 0-1 X_{rt}, em número de $k*\tau$, em que τ é um limite superior em C_{max}, tais que $X_{rt} = 1$ se, e só se, O_r está sendo processada no tempo t. A função objectivo serve para minimzar C_{max}. O número de total de (equações de) constrangimentos é $2*k*\tau+(m-n)*\tau$.

Noutro modelo, que usa uma função objectivo idêntica ([Pritsker 1969], descrito em [Kan 1976]), $X_{rt} = 1$ se, e só se, $s_r+p_r = t$. O número de constrangimentos é igual a $2*k+m*\tau-n$. O modelo é mais compacto que o anterior, embora o número de variáveis 0-1 permaneça grande.

Num modelo de programação inteira mista para o problema de job-shop scheduling apresentado em [Shapiro 1993], um caso com 10 processos de 10 operações cada um e com 8 máquinas dá origem a 2580 restrições, 201 variáveis contínuas e 240 variáveis 0-1. Como é

referido por aquele autor, para contornar a dificuldade computacional na resolução do problema, vários investigadores sugerem outros métodos como combinações de branch and bound (ver mais à frente), métodos heurísticos e limites inferiores para o valor de critério de avaliação de programas obtidos a partir de versões relaxadas/simplificadas, fàcilmente optimizáveis, do modelo do mesmo problema.

Na prática, métodos de programação matemática aplicados ao scheduling, originam frequentemente modelos com um número elevado de variáveis e constrangimentos, mesmo para um problema pequeno, para além de não tirarem partido da estrutura especial dos problemas de scheduling [Kan 1976].

3.1.4. Branch and Bound

Os métodos de branch and bound foram inicialmente desenvolvidos e usados no contexto da resolução de problemas de programação inteira mista [Land 1960] e do problema do caixeiro viajante [Eastman 1959], sendo dos mais usados para resolver problemas de optimização combinatória.

Branch and bound é um método que segue um princípio de *enumeração implícita*, isto é, considera certas soluções possíveis apenas indirectamente e sem as avaliar explicitamente. Um outro princípio subjacente é o de subdividir um problema em subproblemas, em princípio independentes, e mais simples. O método consiste em considerar subconjuntos sucessivamente mais pequenos do conjunto de soluções possíveis (correspondendo a subproblemas do problema original cada vez mais pequenos) até obter conjuntos com uma única solução, ou conjuntos que, por certo, não contêm a solução óptima. Há três aspectos básicos neste método:

- O procedimento de branch (ramificar) - Particionamento de um problema grande em dois ou mais subproblemas.
- O procedimento de bound (limitar) - Calcula um limite inferior da solução óptima para cada subproblema.
- A estratégia de procura usada - Tem a ver com os subproblemas que são escolhidos para tomar em consideração em cada passo.

A execução do procedimento branch pode ser representada por uma árvore de procura, em que o problema original corresponde ao nó da raiz e os subproblemas de um problema aos nós filhos de um nó do nível imediatamente anterior. É mantida uma lista de nós não processados, correspondentes aos subproblemas ainda não eliminados, ou àqueles cujos subproblemas não foram ainda gerados. O modo como esta lista é gerida e escolhido o próximo nó para expandir em nós filhos tem a ver com uma estratégia usada no método. É também memorizado o valor (de critério) para a melhor solução encontrada até ao presente. Este valor serve como *limite superior* para soluções, ou conjuntos de soluções, a encontrar em subproblemas que se encontrarem nos seguintes passos. O procedimento de bound permite calcular um *limite inferior* para o valor (de critério) para cada subproblema. Sempre que este limite é maior que o limite superior não vale a pena explorar o nó correspondente, isto é, não vale a pena gerar os seus subproblemas pois, através deles, as soluções que se iriam encontrar teriam um valor maior (pior, portanto) do que uma já encontrada. Isto significa que um (possivelmente grande) número de soluções possíveis, por certo não óptimas, foi posto de parte.

Uma estratégia possível é a de procura em árvore tipo *depth-first* (em profundidade primeiro) com *backtracking* (retrocesso) em que nós de um ramo da árvore de procura são explorados até ao fim (até gerar uma solução completa), retrocedendo depois para ir expandir nós em níveis anteriores. Uma estratégia alternativa é a de *jumptracking*, ou procura em fronteira, que mantém uma fila de nós ainda não processados, escolhendo em cada passo aquele que tiver menor limite inferior. Esta útlima estratégia origina uma procura muito semelhante à do algoritmo conhecido por A* e frequentemente referido na literatura de Inteligência Artificial.

A simplicidade do princípio básico do branch and bound, a sua fácil implementação [Lenstra 1975] e eficiência computacional parecem ser razões básicas da sua popularidade. No entanto, pela sua natureza, este tipo de métodos tem um comportamento computacional imprevisível (não há nenhuma maneira de prever, em geral, quantos nós não terão de ser examinados pelo facto de terem associados limites inferiores suficientemente grandes), para além de permanecerem intratáveis, pois o número de passos do algoritmo aumenta numa razão exponencial em relação ao tamanho do problema dado [Blazewicz 1994], [Kan 1976].

3.1.5. Programação Dinâmica

Tal como o branch and bound, o método da programação dinâmica é um de enumeração implícita e de subdivisão do problema em subproblemas. O método foi elaborado nos anos 50 por Bellman (ver [Bellman 1957] e [Bellman 1962]).

Segundo o método da programação dinâmica o problema de scheduling (ou outro problema de optimização) é visto como um processo de decisão em vários estágios. Em cada estágio deve ser tomada uma decisão que tem impacto nas decisões a tomar em estágios subsequentes. O princípio de optimalidade de Bellman (que diz que, começando-se num estágio qualquer, a política óptima para os estágios subsequentes é independente da aplicada nos estágios anteriores) serve para obter uma equação recursiva que descreve o valor óptimo de critério num dado estágio em função do valor prèvio. Este princípio é aplicável a muitos problemas de optimização.

Este método leva à exploração de um grafo de procura, numa forma similar à da árvore de procura do branch and bound, havendo também um limite inferior dado pelos valores de custos até ao estágio actual, e pondo-se de parte também conjuntos de alternativas possíveis.

À semelhança do que foi dito para o branch and bound, também o método da programação dinâmica não é computacionalmente atractivo. O tamanho do grafo de procura é normalmente grande e o mecanismo para pôr de parte conjuntos de soluções possíveis subóptimas pode ser bastante fraco [Kan 1976].

3.1.6. Análise de Compexidade

Pode perceber-se que a complexidade é um factor de peso no problema de scheduling. Efectivamente o problema geral de scheduling é um dos problemas que pertence à classe NP-difícil (ver [Garey 1979]): o número de soluções possíveis (fisicamente realizáveis, que não violam os constrangimentos impostos) é frequentemente grande, mesmo para um problema pequeno mas, para além disso e em particular, esse número aumenta numa razão exponencial em relação à dimensão do problema (os valores dos parâmetros). Estes factores tornam os problemas desta classe intratáveis do ponto de vista computacional, e não há método algorítmico nenhum que os consiga solucionar (encontrar a solução óptima) em tempo útil.

No entanto, dependendo do caso particular de problema poderão existir, ou não, algoritmos de optimização eficientes, quer dizer, algoritmos que encontrem a solução do problema em tempo polinomial relativamente à dimensão do problema. A resposta a esta questão pode ser dada pela análise da complexidade inerente ao caso particular de problema de scheduling. Então, a abordagem de um problema de scheduling deverá começar por uma análise da sua complexidade, e o resultado final desta pode ser um dos seguintes (ver [Blazewicz 1994], Cap.3):

- Sim, o problema pode ser resolvido (determinando-se o programa óptimo) em tempo limitado por uma função polinomial da dimensão do problema. A utilidade do algoritmo apropriado para o caso de problema dependerá do grau da sua função de complexidade no pior caso, ou no caso médio, e da sua aplicação particular.
- Não, o problema é NP-difícil.

Neste último caso, segundo [Blazewicz 1994], poderemos optar por:

- Usar um *método aproximado*, introduzindo *simplificações* no problema original por relaxação de alguns dos constrangimentos nele impostos (p.ex., permitir preempção, mesmo se o problema não é de programação preemptiva; assumir operações com duração unitária, mesmo não sendo o caso; assumir certo tipo de grafos de precedência como árvores, ou cadeias, quando o grafo é qualquer no problema original). Isto equivale a resolver uma versão simplificada do problema real, cuja solução pode ser uma boa aproximação da solução do problema original.
- Usar *métodos aproximados* que fazem uso de *algoritmos heurísticos*. Estes tendem a encontrar um programa óptimo mas sem sucesso garantido. Enquadram-se aqui o algoritmo de programação com *fila de prioridade*, aplicado ao sequenciamento das operações nas máquinas, ou um dos *métodos heurísticos*, descritos mais abaixo.
- Usar *algoritmos exactos* se o problema é resolúvel por um algoritmo de optimização *pseudopolinomial*, cuja função de complexidade no pior caso é limitada por um polinómio do maior valor presente na instância do problema. Para números razoàvelmente pequenos o algoritmo pode mesmo comportar-se bem na prática, podendo mesmo ser usado num programa de computador.

3.2. As Questões Problemáticas

Embora todo o trabalho teórico, de índole matemática, orientado para métodos exactos, tenha trazido vários avanços ao longo dos anos para o problema do scheduling, ele não é completamente bem sucedido em aplicações realistas. A complexidade revela-se um obstáculo com algoritmos de optimização combinatória, salvo em casos em que se introduzam simplificações que, não raro, são tão drásticas que nos afastam do problema real. Este aspecto tem sido referido na literatura clássica e moderna sobre scheduling, por vários investigadores.

Por exemplo, em [Graves 1981], Secção 4, reconhece-se que "... há um distanciamento entre teoria e prática do scheduling da produção" e que "o desenvolvimento de melhores algoritmos e heurísticas é essencial para que a teoria do scheduling possa continuar a melhorar a prática do scheduling". Para além disto, e no que respeita a direcções futuras (da investigação, em 1981) este autor dizia haver uma grande necessidade "... de modelos de scheduling mais realísticos e de uma melhor compreensão da dinâmica inerente ao ambiente de scheduling".

Também segundo [Dorn 1994] há 3 tipos de problemas com os modelos analíticos/formais referidos acima:

- Os algoritmos são demasiado complexos para aplicações do mundo real.
- Os modelos exigem o conhecimento exacto de durações e restrições técnicas.
- O esforço necessário para formalizar um novo problema é considerável.

3.3. Métodos Heurísticos

3.3.1. Introdução

Em face do obstáculo da complexidade passou-se a dar uma maior atenção e importância a métodos aproximados, de tipo heurístico.

Certas heurísticas são tão específicas e dependentes do tipo de problema que só podem ser usadas para um tipo de problema particular. Este é o caso do algoritmo de *fila de prioridade*, descrito na secção seguinte.

Na década de 80, no entanto, a investigação orientou-se para métodos heurísticos que usam heurísticas tão gerais que podem ser usadas numa larga quantidade de problemas, podendo mesmo ser usadas em combinação com outros métodos, daí serem muitas vezes referidos na literatura por *meta-heurísticas* (ver [Reeves 1993a]). Uma característica comum destes métodos é a de convergirem para uma solução que é um óptimo global no espaço do problema, para a maior parte dos casos, mas em que isso não é garantido para algum caso em particular (podendo mesmo arriscarem-se a encontrar soluções subóptimas). Alguns destes métodos heurísticos como *simulated annealing*, *tabu search* e *algoritmos genéticos*, aplicáveis a problemas de optimização combinatória e portanto, também ao scheduling, descrevem-se resumidamente a seguir.

Segundo [Reeves 1993a], sendo as heurísticas em geral mais flexíveis e capazes de tratar funções objectivo e/ou constrangimentos mais complicados, tornam possível termos modelos exactos, ou pelo menos, muito mais aproximados do problema real. Enquanto que com os outros métodos (clássicos) nos é possível, em geral, obter soluções aproximadas de um modelo exacto do problema real (havia um modelo exacto, por exemplo de programação matemática, do problema, mas ao qual era necessário introduzir simplificações), com métodos heurísticos poderemos modelar o mundo real de forma mais precisa (ter um modelo exacto ou, pelo menos, tanto quanto possível) e obter soluções que poderão (ou não) ser aproximadas (ver [Reeves 1993a] (pág. 11).

3.3.2. Fila de Prioridade

Uma método genérico frequentemente usado, consiste no que se designa algoritmo de programação com *fila de prioridade*, ou *list scheduling algorithm* [Blazewicz 1994], também referido como algoritmo de programação com *regras despacho*, ou *regras de prioridade* [Kan 1976] ou *regras heurísticas* [Roldão 1995]. Neste, para cada processador, há uma fila de operações por programar e, em cada passo, é escolhida para processar a primeira operação da fila.

A precisão deste algoritmo aproximado depende da ordem pela qual as operações a programar são colocadas na fila e do critério de optimalidade [Blazewicz 1994]. A ordem das operações a programar na fila é prescrita por uma regra de prioridade.

Várias regras de prioridade podem ser empregues e estudos de simulação demonstram que, embora certas regras sejam, em geral, melhores que outras, nenhuma regra é melhor em qualquer circunstância [Brown 1995], [Kan 1976]. Descrevem-se a seguir algumas destas regras (para maior detalhe e outras regras ver [Kan 1976], [Chase 1995] ou [Roldão 1995]).

- **EDD** (Earliest Due Date) - Regra da *menor data limite*: prioridade maior à operação com menor data limite.
- **STR** (Slack Time Remaining) - Regra da *folga mínima*: prioridade maior à operação com menor folga. A *folga* é a diferença entre o tempo restante antes da data limite e o tempo de processamento restante, isto é, a diferença entre o tempo entre o momento da decisão e a data limite e o tempo de processamento entre o momento de decisão e a data limite.
- **SPT** (Shortest Processing Time) - Regra do *menor tempo de processamento*: prioridade à operação com o tempo de processamento mais curto.
- **FIFO** (First In First Out) - Regra *primeiro a entrar-primeiro a sair*: prioridade maior à operação que primeiro fica disponível para processamento.
- **LIFO** (Last In First Out) - Regra *último a entrar-primeiro a sair*: prioridade maior à última operação que ficou disponível para processamento.
- **RANDOM** - *Sorteio*: prioridade é dada a uma operação escolhida aleatoriamente.
- **CR** (Critical Ratio) - Regra da *menor razão crítica*: prioridade à operação com menor razão crítica. A *razão crítica* é o quociente entre o tempo restante antes da data limite e o tempo de processamento (haverá atraso se CR < 1).
- **NOP** (Número de operações) - Regra do *menor número de operações*: prioridade à operação cujo processo tem menor número de operações por executar.

Experiências feitas em larga escala, segundo [Kan 1976] relatadas em [Conway 1967], indicam que as regras de prioridade funcionam bem na obtenção de programas *non-delay* (programas em que não há nenhuma máquina que fique parada a partir do momento em que inicia o processamento [11]), havendo superioridade das regras SPT e RANDOM.

Segundo [Roldão 1995] a regra SPT funciona bem em ambientes congestionados (quando há processadores críticos, isto é, com muita procura pelas operações) e tem um desempenho óptimo quando não há atrasos; a regra NOP é eficiente quando o número de operações é elevado (devido à maior percentagem de tempos de espera, que são proporcionais ao número de operações); as regras EDD, STR e CR só são boas quando os níveis de carga são baixos; as regras EDD e STR são pouco aplicáveis quando há tempos de espera curtos entre operações e há pouca folga (data limite muito próxima); a conjugação das regras SPT e NOP origina frequentemente bons resultados.

De um modo geral as regras de prioridade são muito especializadas, originando uma optimização muito local do programa, insuficiente em problemas de programação complexos, e poderão não ser apropriadas já que as decisões são tomadas na ordem de implementação

[11] Nestes programas a fragmentação é nula (compacidade máxima) nos programas associados a cada processador.

(são regras de despacho) e não são revogadas (são procedimentos de uma única passagem) [Kan 1976]. Daqui que o programa obtido unicamente através destes métodos não seja, em geral, um programa globalmente óptimo.

3.3.3. Simulated Annealing

Este método baseia-se na noção de vizinhança de uma solução, no espaço de soluções possíveis do problema e pode ser visto como uma variante da técnica heurística de procura local, em que um subconjunto das soluções possíveis é explorado indo repetidamente de uma solução para uma solução vizinha. Trata-se de um tipo de métodos que é frequentemente apelidado de melhoria iterativa por repetidamente procurarem passar de uma solução a uma outra que seja melhor. Em geral, uma solução razoàvelmente boa é encontrada ao fim de algumas iterações. Um caso simples destes métodos é o algoritmo designado por *hill-climbing* (versão de maximização), ou *hill-descending* (versão de minimização), de que o simulated annealing pode ser considerado uma versão aperfeiçoada.

O método de simulated annealing (ver [Kirkpatrick 1983], [Van Laarhoven 1988], [Aarts 1989]) faz uso de uma analogia da Termodinâmica Estatística, no que respeita ao fenómeno da mudança do estado de energia de um sólido quando é submetido a um processo de arrefecimento até convergir para um estado final (estado este que depende do modo como é feito o arrefecimento). Em 1983 Kirkpatrick [Kirkpatrick 1983] propõe usar um algoritmo, proposto em 1953 por Metropolis e outros investigadores para simular aquele fenómeno, para procura de soluções de um problema de optimização.

No algoritmo de Metropolis, cada iteração dá-se para um valor fixo da temperatura do material sólido, começando-se a uma dada temperatura. É gerada uma perturbação no estado do material e calcula-se a alteração de energia correspondente. Se a energia diminuiu o sistema muda para este novo estado, caso contrário o novo estado é aceite mediante o valor de uma dada função de probabilidade. Isto é repetido um número determinado de vezes dentro da mesma iteração (para a mesma temperatura), após o que a temperatura é diminuída, repetindo-se de seguida um novo ciclo até que o material “congele” num estado de final de equilíbrio, de energia mínima. A função de probabilidade é tal que a probabilidade diminui com o aumento da energia e à medida que a temperatura diminui, reflectindo a tendência natural de o material preferir estados de energia mínima, e de essa preferência ser cada vez mais forte à medida que a temperatura diminui.

Na analogia, no problema de optimização, um estado do sólido corresponde a uma solução do problema, o nível de energia do sólido ao valor da função objectivo para uma solução e a temperatura a um parâmetro de controlo que varia (diminui) em cada iteração, servindo bàsicamente de contador de iterações. O estado final corresponde à solução encontrada. O método começa com uma solução subóptima do problema e para um certo valor do parâmetro de controlo (correspondente à temperatura). Em cada iteração este parâmetro mantém-se constante. É escolhida uma solução vizinha da solução actual (aleatòriamente entre as soluções vizinhas) que será aceite como solução actual se o seu custo for menor. Caso contrário (custo maior) a solução escolhida pode mesmo assim ser acite se um valor aleatòriamente gerado for menor que o valor da função de probabilidade (o uso de aleatoriedade leva a que o simulated annealing seja frequentemente referido como um método de Monte Carlo). Isto é repetido um número determinado de vezes dentro da mesma iteração, após o que o parâmetro de controlo é decrementado, repetindo-se de seguida um novo ciclo até que o parâmetro de controlo seja zero.

Um dos problemas com os algoritmos de melhoria iterativa é o de se poderem encaminhar para soluções que correspondem a pontos óptimos (mínimos) locais, e não globais, da função de custo. Correm assim o risco de terminarem com soluções que não são óptimas e este é o problema do algoritmo, acima referido, de hill-descending. É isto que o simulated annealing tenta evitar, ao permitir aceitar encaminhar-se para soluções piores do que a actual, mas mais em iterações iniciais, onde o valor da função de probabilidade é mais alto. À medida que o parâmetro da temperatura diminui a liberdade para mover-se para uma solução pior diminui, de modo que o comportamento esperado do algoritmo será o de dar inicialmente alguns "saltos" entre pontos afastados no espaço de soluções possíveis, em princípio obtendo uma solução próxima do óptimo, concentrando-se de seguida na exploração da vizinhança dessa solução até obter a solução óptima. Para isto, no entanto, é necessário afinar certos parâmetros, nomeadamente o perfil de evolução da temperatura (taxa a que o parâmetro da temperatura diminui) - valores inicial e final, número de repetições para cada temperatura e decréscimo desta de iteração para iteração. Para cada tipo particular de problema haverá ainda que definir a função de custo e a vizinhança (ver [Dowsland 1993]).

Para além das vantagens já descritas para os métodos heurísticos, os métodos como o simulated annealing têm como vantagens serem computacionalmente económicos em termos de espaço de memória necessário, pois mantêm em memória apenas o estado actual (não "olham" mais à frente do que os estados sucessores dele e não mantêm estados antecessores), o que é vantajoso em especial em espaços de solução densos. Uma grande desvantagem é a de serem demorados (consomem muito tempo) e de haver necessidade de afinação dos parâmetros. Pouca aplicação terão em situações em que o caminho pelo qual a solução é atingido é relevante (já que não mantêm informação de estados antecessores).

Grande parte das aplicações deste método referidas no campo da Investigação Operacional envolvem problemas de scheduling e geração de horários [Dowsland 1993]. Muitas das aplicações ao scheduling têm a ver com a determinação da sequência óptima para um certo número de processos num dado conjunto de máquinas de modo a minimzar o tempo total de produção (comprimento do programa). O espaço de soluções é constituído por todas as sequências possíveis e podem ser definidas uma série de tipos de vizinhança, como as que se obtêm por mudança da posição de um processo, ou troca das posições de dois processos. Na geração de horários a vizinhança de uma solução é obtida por troca da afectação de dois eventos (ou aulas, ou sessões) em tempos diferentes, ou cancelando um e substituindo-o. A função de custo mede a não aceitabilidade de uma solução podendo envolver uma soma ponderada de penalidades associadas, por exemplo, a vários tipos de violações de constrangimentos, ou outros factores não aceitáveis, em maior ou menor grau, num horário.

[Dowsland 1993] descreve resumidamente aplicações com sucesso do simulated annealing à sequenciação no problema de flow-shop ([Ogbu 1990], [Ogbu 1991], [Osman 1989]), problemas de produção de peças envolvendo vários estágios com minimzação de custos de preparação e de existências ([Kuik 1990]), programação de sistemas de produção flexível, em que é tido em conta, para além dos processos e das máquinas, também o sistema de transporte entre as máquinas ([Brandimarte 1987]), programação dos transportes ([Wright 1989]) e geração de horários ([Abramson 1991], [Eglese 1987], [Dowsland 1990]).

3.3.4. Tabu Search

À semelhança do simulated annealing também este método faz uso da noção de vizinhança e de uma analogia com um fenómeno natural que é, no caso, o uso de uma forma de memória.

O princípio básico do tabu search (ver [Glover 1986], [Hansen 1986], [Glover 1993]) é o de uma procura heurística do tipo hill-descending, guiada para explorar o espaço de soluções de um modo a não se encaminhar para soluções localmente óptimas que já tenha prèviamente encontrado. Para isto é usada uma estrutura de memória designada por *lista tabu*, que contém em cada momento um número de soluções proibidas determinadas por uma história relativa a um certo número de iterações anteriores. Alternativamente, a lista tabu pode memorizar, não as próprias soluções proibidas, mas os movimentos no espaço de soluções que levam a elas. Esta forma alternativa pode economizar memória para armazenamento da lista tabu e economizar tempo de procura nela, mas poderá revelar-se demasiadamente restritivo proibindo soluções para as quais não havia essa intenção.

No método de tabu search em cada iteração passa-se da solução actual para uma na vizinhança que não esteja na lista tabu e que mais melhore o valor da função objectivo, ou a que menos o piora, se não houver nenhuma melhor que a actual. Para além disto, existe um *critério de aspiração* que determina se um movimento é admissível apesar de figurar na lista tabu. Podem existir, no contexto deste método, estratégias de *intensificação*, para concentrar a procura em regiões do espaço de soluções mais promissoras ou de *diversificação*, para conduzi-la para regiões não exploradas.

Outras características são o uso de *listas de candidatos*, usadas para reduzir o número de movimentos investigados em cada iteração, quando a vizinhança, ou o custo computacional de avaliar cada solução, são muito grandes; *soluções elite*, que são soluções boas (óptimos locais, ou próximas do óptimo global) que são usadas para guiar uma estratégia de intensificação; *frequência* de aparecimento de certos elementos na solução para guiar estratégias de diversificação [Ribeiro 1996].

Este método requer o uso intensivo da memória e uma implementação muito cuidada (excessivo número de parâmetros a estabelecer) mas oferece as vantagens de poder produzir boas soluções em tempos relativamente pequenos, se for bem aplicado [Ribeiro 1996].

[Glover 1993] descreve resumidamente várias aplicações com sucesso do tabu search a variados problemas de scheduling.

3.3.5. Algoritmos Genéticos

Este método faz uso de uma analogia com o fenómeno natural da evolução dos organismos vivos. A ideia base dos Algoritmos Genéticos (ver [Holland 1975], [Goldberg 1989], [Reeves 1993b]) é a de simular o desenvolvimento de populações de indivíduos de uma espécie com características cada vez melhores, ou mais desejáveis, face a uma selecção natural. Estas características são determinadas ao nível genético pelo modo como são combinados os cromossomas dos progenitores.

O algoritmo começa com uma população inicial de um certo número de cromossomas, progenitores potenciais, cujos valores de qualidade foram determinados. O valor de qualidade corresponde ao valor de uma *função de avaliação* usada para medir o grau de aptidão de um cromossoma. Há depois uma *selecção* dos cromossomas de maior qualidade para darem origem a uma nova geração e aplicam-se-lhes *operadores genéticos* para obtenção dos cromossomas da geração seguinte. Estes operadores podem ser o *cruzamento*, ou a *mutação*. O primeiro consiste na troca de genes de dois cromossomas progenitores; o segundo na alteração de genes de um cromossoma. O processo é repetido até uma certa *condição de terminação* se verificar. Esta condição pode ser por exemplo, ter decorrido um certo número

de iterações/gerações, ou o melhor valor de qualidade na população ser constante ao longo de um certo número de gerações, ou a população ser dominada por um certo número de indivíduos. Este método pode ser visto como uma procura aleatória no espaço de soluções possíveis, conduzida bàsicamente pelos operadores genéticos que são aplicados e pela forma de selecção.

Os cromossomas são representados por uma string de dígitos binários (0s e 1s), ou números. Factores a considerar num algoritmo genético são o tamanho da população, a (geração da) população inicial, os mecanismos de selecção, a função de avaliação, e os operadores genéticos.

Ao usar um algoritmo genético com um problema de optimização, cada cromossoma representa uma solução possível e os genes os elementos (ou variáveis) da solução. O valor de qualidade corresponde ao valor da função objectivo.

[Reeves 1993b] relata aplicações dos algoritmos genéticos à sequenciação e ao scheduling em especial à sequenciação no problema de flow-shop, ao problema do scheduling estocástico de flow-shop e ao problema de job-shop. No primeiro caso fizeram-se comparações da perfomance do algoritmo genético implementado com a de uma heurística tipo simulated annealing, altamente eficiente, concluindo-se por uma equiparação dos dois métodos em geral, mas uma perfomance crescente do algorimo genético à medida que a dimensão do problema aumentava. Também outro estudo relatado conclui por uma perfomance equiparável à do método de tabu search.

3.4. Aspectos Críticos

Os métodos heurísticos descritos apresentam realmente vantangens, nomeadamente a de serem tão gerais que são aplicáveis numa larga quantidade de problemas, poderem ser usados em combinação com outros métodos, tornarem possível modelos muito mais aproximados do problema real (por serem flexíveis e capazes de tratar funções objectivo e/ou constrangimentos mais complicados) e serem de fácil implementação. Como foi dito é uma característica comum a de tenderem a convergir para uma solução que é um óptimo global no espaço do problema, embora isso não seja garantido para algum caso em particular, arriscando-se a encontrarem soluções subóptimas. Esta propriedade de convergência parece ser importante na medida em que permite, pelo menos, a melhoria, ou reparação, iterativa de uma solução subóptima. Além disso estes algoritmos dispõem de, e podem ser terminados com, uma solução ao fim de uma qualquer iteração (ainda que seja uma solução subóptima) e que seria melhorada na próxima iteração, uma propriedade que leva a serem apelidados de *algoritmos anytime*, na literatura moderna. O facto de trabalharem sobre uma solução completa parece ter a vantagem de permitir uma avaliação (através da função objectivo) mais precisa da solução actual. Este problema põe-se quando o método usado não funciona por construção, ao invés de por reparação, isto é, constrói a solução por junção progressiva dos seus componentes (o problema põe-se no branch and bound, na programação dinâmica, num algoritmo com lista de prioridade, ou noutro método qualquer que em cada passo dispõe de uma solução parcial, ou de um conjunto de soluções parciais que é necessário avaliar para conduzir a procura nos passos subsequentes).

Um aspecto que, no entanto, penaliza fortemente estes algoritmos é o de não serem eficientes. Mesmo se abandonarmos o objectivo de encontrar uma solução óptima e nos contentarmos com uma solução satsifatória para um problema, o que pode ser feito de um modo muito fácil (terminando o algoritmo quando a solução já ultrapassou um nível de qualidade considerado

aceitável), a obtenção de uma solução satisfatória pode ser demorada. Por outras palavras, não serão apropriados para scheduling reactivo, isto é, scheduling com um horizonte temporal curto em que pode haver a necessidade de rever a programação de modo a reagir a eventos inesperados, como falhas de máquinas, ou outros recursos, variações nas durações das operações, chegada de processos novos para execução, cancelamento de processos, com um número de disrupções e prejuízo na qualidade mínimos no programa.

Um outro aspecto que parece crítico, se aumentarmos o nosso nível de exigência, é o de, por si só, não permitirem um modelo do problema que possa ser suficientemente próximo do problema real de modo a tirar partido da sua estrutura particular. Não preconizam um quadro geral onde se enquadrem os aspectos de modelação e representação explícita de conhecimento e de aspectos/entidades do domínio como máquinas, recursos, processos e operações, programas, ambientes de produção, constrangimentos de vários tipos, resolução distribuída/descentralizada do problema, etc.. Dadas as exigências colocadas ao scheduling actual na produção, nos transportes, nos serviços, uma maior aproximação ao real faria sentido. Ou, indo mais longe (e sendo mais exigente), faria sentido existir uma forma de modelar o problema do scheduling suficientemente abrangente e geral, mas que permita abordar cada problema de scheduling específico tirando inclusive partido da sua estrutura particular.

4. REFERÊNCIAS

Aarts 1989 — Aarts, E.H.L.; Korst, J.H.M., *Simulated Annealing and Boltzman Machines*, Wiley, Chichester, 1989.

Abramson 1991 — Abramson, D., Constructing *School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms*, Management Science, 37 1991, 98-113.

Baker 1974 — Baker, K.R., *Introduction to Sequencing and Scheduling*, New York, Wiley, 1974.

Bellman 1957 — Bellman, R., *Dynamic Programming*, Princeton University Press, Priceton, New Jersey, 1957.

Bellman 1962 — Bellman, R,; Dreyfus, S.E., *Applied Dynamic Programming*, Princeton University Press, Priceton, New Jersey, 1962.

Blazewicz 1994 — Blazewicz, J.; Ecker, K.H.; Schmidt, G.; Weglarz, J., *Scheduling in Computer and Manufacturing Systems*, Springer Verlag, 1994.

Bowman 1959 — Bowman, E.H., *The Schedule-Sequence Problem*, Operations Research, vol. 7, pp. 621-624, 1959.

Brandimarte 1987 — Brandimarte, P.; Conterno, R.; Laface, P., *FMS Production Scheduling by Simulated Annealing*, Proceedings of the third International Conference on Simulation in Manufacturing, 235-245.

Brown 1995 — Brown, Donald E.; Marin, John A.; Scherer, William T., *A Survey of Intelligent Scheduling Systems*, in: Intelligent Scheduling Systems, Brown, D.E.; Scherer, W.T. (eds.), Kluwer Academic Publications, pp. 1-40, 1995.

Burke 1994 — Burke, P.; Prosser, P., *The Distributed Asynchronous Scheduler*, in: Intelligent Scheduling, Zweben, Monte; Fox, Mark S. (eds.), Cap.1, San Francisco, Morgan Kaufman, 1994.

Chase 1995 — Chase, Richard B.; Aquilano, Nicholas J., *Gestão da Produção e das Operações, Perspectiva do Ciclo de Vida*, Monitor, Lisboa, 1995.

Dorn 1994 — Dorn, Jurgen; Slany, Wolfgang, *A Flow Shop with Compatibility Constraints in a Steelmaking Plant*, in: Intelligent Scheduling, Zweben, Monte; Fox, Mark S. (eds.), Cap.22, San Francisco, Morgan Kaufman, 1994.

Dowsland 1990 — Dowsland, K.A., *A Timetabling Problem in which Clashes Are Inevitable*, JORS, 41 1990, 907-918.

Dowsland 1993 — Dowsland, K.A., Chapter 2- *Simulated Annealing*, in: Reeves, C.R. (Ed.), in: Modern Heuristic Techniques for Combinatorial Problems, Reeves, C.R. (ed.), Blackwell Scientific Publications, 1993.

Eastman 1959 — Eastman, W.L., *A Solution to the Travelling Salesman Problem*, Econometrica, vol. 27 1959, 282.

Eglese 1987 — Eglese, R.W.; Rand, G.K., *Conference Seminar Timetabling*, JORS, 38 1987, 591-598.

Fox 1994 — Fox, Mark S., *ISIS: A Retrospective* in: Intelligent Scheduling, Cap.1, Zweben, Monte; Fox, Mark S., (eds.), San Francisco, Morgan Kaufman, 1994.

Froeschl 1993 — Froeschl, Karl A., *Two Paradigms of Combinatorial Production Scheduling* in: Scheduling of Production Processes, Dorn, Jurgen; Froeschl, Karl, (eds.), Elis Horwood Limited, 1993.

Garey 1979 — Garey, M.R.; Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.

Glover 1986 — Glover, F., *Future Paths for Integer Programming and Links to Artificial Intelligence*, Computers and Operations Research, 5 1986, 533-549.

Glover 1993 — Glover, F., Chapter 3- *Tabu Search*, in: Modern Heuristic Techniques for Combinatorial Problems, Reeves, C.R. (ed.), Blackwell Scientific Publications, 1993.

Goldberg 1989 — Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*,

Addison-Wesley, Reading, Massachussets, 1989.

Graves 1981 — Graves, Stephen C., *A Review of Production Scheduling*, Operations Research 28 1981, 646-675.

Hansen 1986 — Hansen, P., *The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming*, Congress on Numerical Methods in: Combinatorial Optimization, Capri, Italy, 1986.

Holland 1975 — Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

Kan 1976 — Rinnooy Kan, A.H.G., *Machine Scheduling Problems, Classification, Complexity and Computations*, Martinus Nijhoff, The Hague, 1976.

Kirkpatrick 1983 — Kirkpatrick, S.; Gellat, C.D.; Vecchi, M.P., *Optimization by Simulated Annealing*, Science, 220 1983, 671-680.

Kuik 1990 — Kuik, R.; Salomon, M., *Multi-Level Lot-Sizing Problem: Evaluation of a Simulated Annealing Approach*, EJOR, 45 1990, 25-37.

Land 1960 — Land, A.H.; Doig, A.G., *An Automatic Method for Solving Discrete Programming Problems*, Econometrica, vol. 28 1960, 497-520.

Lawler 1989 — Lawler, E.L.; Lenstra, J.K.;Rinnooy Kan, A.H.G.;Shmoys, D.B., *Sequencing and Scheduling: Algorithms and Complexity*, Report BS-R8909, Centre for Mathematics and Computer Science - Dep. of Oper. Research, Statisctics and System Theory, 1989.

Le Pape 1994 — Le Pape, Claude, *Scheduling as Intelligent Control of Decision-Making and Constraint Propagation*, in: Intelligent Scheduling, Cap.3, Zweben, Monte; Fox, Mark S. (eds.), San Francisco, Morgan Kaufman, 1994.

Lenstra 1975 — Lenstra, J.K.; Rinnooy Kan, A.H.G., *A Recursive Approach to the Generation of Conbinatorial Configurations*, Working Paper WP/75/25, Graduate School of Management, Delft, 1975.

McDermott 1995 — McDermott, Drew; Hendler, James, *Planning: What it is, What it could be, An Introduction to the Special Issue on Planning and Scheduling*, Artificial Intelligence, 76 (1995), 1-16.

Muscettola 1994 — Muscettola, Nicola, *HSTS: Integrating Planning and Scheduling*, in: Intelligent Scheduling, Cap.6, Zweben, Monte; Fox, Mark S. (eds.), San Francisco, Morgan Kaufman, 1994.

Ogbu 1990 — Ogbu, F.A.; Smith, D.K., *The Application of the Simulated Annealing Algorithm to the Solution of the n/m/Cmax Flowshop Problem*, Computers & Operations Research, 17 1990, 243-253.

Ogbu 1991 — Ogbu, F.A.; Smith, D.K., *Simulated Annealing for the Permutation Flow-Shop Scheduling*, OMEGA, 19 1991, 65-67.

Osman 1989 — Osman, L.H.; Potts, C.N., *Simulated Annealing for the Permutation Flow-Shop Scheduling*, OMEGA, 17 1989, 551-557.

Pritsker 1969 — Pritsker, A.A.B.; Watters, L.J.; Wolfe, P.M., *Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach*, Management Science, vol. 16, pp. 93-108, 1969.

Reeves 1993a — Reeves, C.R.; Beasley, J.E., Chapter 1- *Introduction*, in: Modern Heuristic Techniques for Combinatorial Problems, Reeves, C.R. (ed.), Blackwell Scientific Publications, 1993.

Reeves 1993b — Reeves, C.R., Chapter 4- *Genetic Algorithms*, in: Modern Heuristic Techniques for Combinatorial Problems, Reeves, C.R. (ed.), Blackwell Scientific Publications, 1993.

Ribeiro 1996 — Ribeiro, C., *Meta-Heuristics and Applications*, Chapter 4.3, Notas do EAIA-96, Escola Avançada de Inteligência Artificial 1996, Estoril, Portugal, 1996.

Roldão 1995 — Roldão, Victor Sequeira, *Planeamento e Programação da Produção*, Ed. Monitor, Lisboa, 1995.

Sadeh 1994 — Sadeh, Norman, *Micro-Oportunistic Scheduling: The Micro-Boss Factory Scheduler*, in: Intelligent Scheduling, Cap.4, Zweben, Monte; Fox, Mark S. (eds.), San Francisco, Morgan Kaufman, 1994.

Shapiro 1993 Shapiro, Jeremy F., *Mathematical Programming Models and Methods for Production Planning and Scheduling*, in: Handbooks in OR & MS, Graves, S.C. et al. (ds.), Vol. 4, Cap. 8, Elsevier Publishers B.V., 1993.

Smith 1994 Smith, Stephen, *OPIS: A Methodology and Architecture for Reactive Scheduling*, in: Intelligent Scheduling, Cap.2, Zweben, Monte; Fox, Mark S. (eds.), San Francisco, Morgan Kaufman 1994.

Van Laarhoven 1988 Van Laarhoven, P.J.M.; Aarts, E.H.L., *Simulated Annealing: Theory and Applications*, Kluwer, Dordrecht, 1988.

Wright 1989 Wright, M.B., *Applying Stochastic Algorithms to a Locomotive Scheduling Problem*, JORS, 38 1989, 187-192.

5. OUTRAS FONTES

Reis 1995 Reis, Joaquim, *Gestão de Recursos e Tempo Usando Técnicas de Inteligência Artificial*, projecto de tese de doutoramento, I.S.C.T.E., Lisboa, 25-Jan-1995.