Instituto Superior de Ciências do Trabalho e da Empresa

O SCHEDULING em INTELIGÊNCIA ARTIFICIAL

um RESUMO

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

Novembro de 1996

1. Introdução

O objectivo deste texto é descrever resumidamente algumas metodologias usadas em sistemas de Scheduling automatizado originados da investigação e desenvolvimento no campo da Inteligência Artificial, principalmente na última década.

Não sendo intenção rever, analisar, ou classificar, em detalhe e exaustivamente estas metodologias, bem como os casos de aplicação, a escolha das mesmas foi mais feita numa base de aplicabilidade prática, popularidade e originalidade das soluções apresentadas para o problema do Scheduling. Para não tornar o texto extenso não se inclui aqui uma definição formal do problema e uma exposição da terminologia e notação tipicamente usadas. Esse aspecto foi focado num outro texto (ver [Reis 1996]) onde se faz uma introdução ao scheduling clássico e uma descrição resumida de métodos de solução clássicos e modernos.

2. O Problema do Scheduling

2.1. Contexto

O Scheduling, ou Programação, ou Programação das Operações, ou Calendarização [1], tem como objectivo a afectação de recursos no tempo necessários para executar um conjunto de processos [Baker 1974]. É uma actividade de gestão de grande importância, por ser um ponto central na Economia o uso criterioso de recursos e do tempo e haver assim necessidade de técnicas de gestão das operações eficientes, tanto na produção, como nos serviços.

É por certo na área da produção industrial que a importância do Scheduling tem maior realce. Numa actividade produtiva o Scheduling está directamente ligado ao Planeamento e à Gestão de Stocks. É, no entanto, disjunto da Gestão dos Stocks, pois as decisões de afectação de recursos de produção são o foco principal dos problemas de scheduling e, por outro, lado são tomadas fora do âmbito da gestão de stocks (que se preocupa com a segurança relativamente às variações na procura); 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 (de modo a poder satisfazer uma procura prevista), e num horizonte temporal relativamente mais dilatado, decisão essa que é exógena ao Scheduling [Graves 1981], [Roldão 1995].

Devido à importância mencionada têm sido modernamente desenvolvidas sistemas computorizados para um uso mais 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 de grandes 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.

2.2. Definição

2.2.1. Definições do Scheduling

Apresentam-se algumas definições (não formais) 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].

---

[1]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, poderão ser 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 1994a].

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].

2.2.2. 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 (se a houver), 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. Dispõe-se de uma função que permite avaliar a qualidade de um programa/solução e pretende-se normalmente obter um para o qual esse valor é óptimo.

A terminologia usada parece frequentemente sugerir que o problema ocorre num contexto de produção industrial, mas na realidade ele ocorre em contextos variados. E é fácil verificá-lo se trocar-mos 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.

3. Soluções Modernas para um Problema Antigo

3.1. Introdução

Uma introdução resumida aos métodos clássicos (programação matemática, branch and bound, programação dinâmica) e a alguns métodos heurísticos modernos (as *meta-heurísticas* simulated annealing, tabu search, algoritmos genéticos) de abordagem do problema foi já feita num outro texto (ver [Reis 1996]).

Recordam-se aqui alguns aspectos problemáticos, focados naquele texto, associados ao problema do scheduling e à sua resolução:

- Complexidade inerente aos problemas de scheduling - O problema geral (de optimização) de scheduling é um dos problemas que pertence à classe NP-difícil (ver [Garey 1979]): o número de soluções possíveis pode ser muito grande, mesmo para um problema pequeno e, para além disso, esse número aumenta numa razão exponencial em relação à dimensão do problema. Estes factores tornam os problemas desta classe intratáveis do ponto de vista computacional, não havendo método algorítmico nenhum que os consiga solucionar (solucionar no caso do problema de scheduling significa encontrar a solução óptima) em tempo útil.
- Métodos de solução - Ao formular o problema de scheduling como um problema de optimização combinatória e procurar encontrar de uma forma exacta o programa óptimo, haverá que enfrentar o obstáculo da complexidade, contornável em casos muito simples, ou se se introduzirem simplificações que, não raro, são tão drásticas que nos afastam do problema real. O trabalho de índole matemática e de desenvolvimentos de modelos analíticos/formais trouxe vários avanços ao longo dos anos para o problema do scheduling, mas ele não é assim, completamente bem sucedido em aplicações realistas. Os modelos são muito complexos e exigem o conhecimento exacto de durações e restrições técnicas, os algoritmos são demasiado complexos para aplicações do mundo real e o esforço necessário para formalizar um novo problema é considerável [Dorn 1994].

Uma tentativa de contornar estes aspectos é o encaminhamento, na década de 80, por métodos heurísticos, nomeadamente o uso de *meta-heurísticas*. Para além de serem de fácil implementação, uma outra característica comum aos algoritmos preconizados por estes métodos é a de convergirem para uma solução que é um óptimo global no espaço de soluções 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). Como as heurísiticas são, 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 1993]).

Alguns aspectos críticos, no entanto, não resolvidos, são:

- Reactividade - Um aspecto que 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. Não são portanto, apropriados para scheduling reactivo, isto é, scheduling em que pode haver a necessidade de rever a programação de modo a reagir a eventos inesperados num horizonte temporal curto, como falhas de máquinas, ou outros recursos, varições nas durações das operações, chegada de processos novos para execução, cancelamento de processos, com um mínimo número de disrupções e prejuízo na qualidade no programa (o aspecto dinâmico/não determinístico do problema não pode ser contemplado).
- Quadro geral de modelação/representação - Aspectos de modelação e representação explícita de conhecimento e de aspectos/entidades do domínio, como tipos de processadores, tipos e categorias de recursos, processos e operações, programas, ambientes de produção, constrangimentos de vários tipos, resolução distribuída/descentralizada do problema, etc. deveriam ser tomados em conta num quadro geral que permita 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.

3.2. O Scheduling em Inteligência Artificial

A Inteligência Artificial é uma área de investigação que procura estudar e reproduzir o comportamento inteligente e que usa tipicamente o computador como ferramenta para testar e implementar teorias. Tendo surgido nos anos 50 e abordando inicialmente problemas relativamente simples (jogos, problemas brinquedo) a sua aplicabilidade prática foi fértil a partir dos anos 70 como o aparecimento dos sistemas periciais, que empregam o conhecimento de um perito num domínio específico para automatizar, ou assistir e apoiar, o desempenho de uma tarefa altamente especializada desse domínio. Factores importantes e que formaram uma base de desenvolvimento desta área foram o processamento simbólico (não numérico) nos computadores e o raciocínio heurístico (muito popularizado por investigadores como Herbert Simon). Na Inteligência Artificial é típica a noção de procura heurística em espaço de estados, de heurísticas e de conhecimento do domínio dos problemas.

Alguns campos de investigação da área de Inteligência Artificial que merecem destaque são a Resolução de Problemas e a Procura Heurística, Representação de Conhecimento, Raciocínio (inclusive Raciocínio que faz uso de conhecimento incompleto, ou Raciocínio Baseado em Casos), Planeamento, Aprendizagem, Compreensão e Geração de Língua Natural, Sistemas Periciais, Visão e Robótica. Um dos campos da Inteligência Artificial é o Planeamento cuja investigação remonta à década de 70. No Planeamento o objectivo é a selecção e sequenciação de acções, formando um plano, de modo a que atinjam um ou mais objectivos e satisfaçam um conjunto de constrangimentos (aplicações na robótica, no planeamento de projecto de experiências científicas, de observações de telescópios automáticos, de produção em job-shop, de montagem). O problema do Scheduling pode ser entendido como uma forma de Planeamento a um nível mais detalhado (ver em 1.2.1 as definições de Mark S. Fox, Nicola Muscettola, e principalmente McDermott & Hendler), em que há que afectar recursos e associar tempos às actividades de um plano, dentro do horizonte temporal a que ele se

estende. Em especial a partir do meio da década de 80 iniciou-se uma fase de investigação dos problemas de Scheduling na área de Inteligência Artificial.

Segundo [Dorn 1994] os sistemas inteligentes, podem oferecer soluções para problemas como o Scheduling, pois:

- Preconizam e facilitam a utilização de heurísticas para reduzir a complexidade do problema.
- Permitem raciocínio com conhecimento incompleto, ou incerto, ou vago, ou probabilístico.
- Representam o conhecimento explicitamente sendo mais fácil desenvolvê-lo e mantê-lo.

Também outro investigador da área refere que, dada a complexidade e diversidade dos problemas de scheduling é natural esperar que procedimentos de programação rígidos, orientados para produzir programas óptimos, em circunstâncias particulares não possam dar, em geral, resultados satisfatórios, noutras circunstâncias [Le Pape 1994]. Ainda segundo o mesmo investigador, uma reacção natural será então, a de usar técnicas de Inteligência Artificial no Scheduling, pelas seguintes razões:

1. Representação e modularidade do conhecimento - Aquelas técnicas facilitam expressão e aplicação do conhecimento do perito da área juntamente com o conhecimento empírico, ou teórico, de outras áreas (ex.: Matemática, Investigação Operacional). Também a distinção entre conhecimento do domínio e o conhecimento de controlo, na resolução do problema, torna mais fácil a adaptação a ambientes diferentes.

2. Análise do problema - Aquelas técnicas são úteis na análise e determinação de qual o conjunto de conhecimentos e quais os componentes do problema que são mais relevantes numa dada situação (ex.: ordens de produção e recursos críticos, efeitos de eventos não esperados, falhas na resolução do problema).

3. Interação com utilizador - Aquelas técnicas facilitam gestão das interações com o utilizador na fábrica, permitindo que ele compreenda as decisões do sistema e possa aceitá-las, ou não.

Não sendo raro que os sistemas de Scheduling actuais usem uma combinação de metodologias oriundas de diferentes áreas (por exemplo, regras de prioridade, recursos congestionados, programação com constrangimentos são comuns à Investigação Operacional [Blazewicz 1994]), pode dizer-se que os que fazem uso mais directo de técnicas de Inteligência Artificial se orientam mais para o uso de conhecimento domínio e heurísticas, e não tanto da ideia de optimização, mas mais da de satisfação. Baseiam-se frequentemente na *satisfação de constrangimentos*: na impossibilidade de encontrar, ou na falta de, uma solução óptima para o problema (problemas de programação há que não têm solução por serem superconstrangidos) tentarão encontrar uma solução que seja satisfatória [Blazewicz 1994]. Para além de constrangimentos numéricos, consideram-se também constrangimentos não numéricos (por exemplo, constrangimentos envolvendo relações entre valores de variáveis em vez de valores simples, ou domínios de valores) [Blazewicz 1994], [Kumar 1992], [Allen 1983], [Ladkin 1988].

O problema de Scheduling é entendido como um *problema de satisfação de constrangimentos* (ou CSP - Constraint Satisfaction Problem). Este é o problema de encontrar valores (recursos e/ou intervalos de tempo) para as variáveis (operações) de um conjunto de variáveis $\mathbf{X} = \{ x_1, x_2, ..., x_n \}$ de tal modo que uma colecção **C** de constrangimentos $c_1, c_2, ..., c_m$ seja satisfeita. A cada variável x_i está associado um domínio d_i que indica os valores possíveis para x_i. Cada constrangimento é um subconjunto do produto cartesiano $d_1 * d_2 * ...$

* d_n que especifica condições para os valores das variáveis x_1, x_2, ..., x_n. Um subconjunto **Y** deste produto cartesiano é uma solução realizável do problema de satisfação de constrangimentos se satisfaz todos os constrangimentos, isto é $\mathbf{Y} \subseteq \cap_{j=1,m} c_j$ [Blazewicz 1994]. Este problema é NP-completo [Garey 1979].

São distinguidos os *constrangimentos rígidos*, ou restrições (os que, ao não serem satisfeitos pela solução, condicionam a sua realizabilidade) dos *constrangimentos relaxáveis*, ou preferências (os que poderão ser, ou não, satisfeitos pela solução mas que, não condicionando a sua realizabilidade, ao serem satisfeitos originam soluções de melhor qualidade). A análise de um problema de satisfação de constrangimentos pode levar a soluções realizáveis, ou à constatação de que não existe solução, para dado conjunto de constrangimentos. Nesse caso podem usar-se técnicas de *resolução de conflitos* como a *relaxação de constrangimentos*, que tentam determinar quais os constrangimentos que podem não ser satisfeitos e/ou qual o grau de não satisfação destes, de modo a obter uma solução satisfatória prejudicando minimamente a qualidade do programa.

A maior parte dos sistemas de Inteligência Artificial constrói uma árvore de procura e aplica uma técnica de *procura guiada por constrangimentos*, para encontrar uma solução realizável. Esta técnica pode variar entre a técnica do *retrocesso* (backtracking simples, ou selectivo, ao detectar um caminho sem saída) e a técnica de *propagação de constrangimentos* (mais sofisticada), cuja idéia básica é a de podar o espaço de procura antes de serem geradas combinações de valores de variáveis que levam a soluções não realizáveis [Blazewicz 1994], ou então uma combinação das duas [Kumar 1992].

De um modo geral estes sistemas dão particular ênfase ao papel da *representação do conhecimento específico do domínio do problema* e decompõem o problema inicial de acordo com variadas perspectivas, como recursos congestionados, hierarquias de constrangimentos, subconjuntos de constrangimentos em conflito. Fazem distinção entre a *representação do conhecimento* (modelos) e a *metodologia* de Scheduling (algoritmos). O conhecimento é aqui empregue para representar entidades do sistema (de produção, por exemplo), constrangimentos, objectivos e as técnicas de representação possíveis podem ser *redes semânticas*, *enquadramentos* (frames), *lógica* ou *regras*. A metodologia é frequentemente baseada em *regras*, *procura heurística* (para guiar a aplicação das regras), *raciocínio oportunístico* (quer dizer, dependendo dos aspectos mais críticos durante a resolução do problema empregar diferentes estratégias e perspectivas de resolução do problema - como por exemplo perspectivas baseadas em recursos, ou baseadas em processos), *decomposição hierárquica* (resolução de subproblemas, abstracção e resolução distribuída de problemas), *emparelhamento de padrões* (por exemplo, uso de informação sobre o estado do sistema de produção e dos objectivos para aplicar regras de prioridade), *propagação de constrangimentos*, técnicas de *imposição* ou *relaxação de constrangimentos* [Blazewicz 1994], [Atabakhsh 1991].

4. Os Sistemas

Descrevem-se a seguir as características mais salientes de alguns sistemas de Scheduling que usam técnicas de Inteligência Artificial mais conhecidos, desenvolvidos entre 1980 e a actualidade.

4.1. ISIS

O projecto ISIS [Fox 1983], [Fox 1994] teve início em 1980 (origens do grupo de investigação envolvido remontam aos anos 60), na Universidade de Carnegie Mellon. O sistema ISIS foi um dos primeiros em que houve a motivação concreta para a aplicação de técnicas de *modelação* e de *procura heurística* de Inteligência Artificial ao problema do Scheduling, no caso, um problema de *job-shop* de uma *fábrica de componentes para turbinas* (Westinghouse). Com o ISIS, pela primeira vez, tentou-se tratar com a gama completa dos constrangimentos e objectivos do domínio da produção.

Mais do que um sistema único, identificou-se no projecto ISIS *uma família de sistemas* de Scheduling automático, baseados em *satisfação de constrangimentos*, cujo objectivo era o de serem desenvolvidos e testados em fases sucessivas, com uma arquitectura progressivamente mais sofisticada em cada fase. Em princípio poderia vir a incorporar-se toda a gama de constrangimentos de situações reais do problema, mas a continuação destes esforços parece ter sido abandonada, embora depois, de alguma forma, continuada pelo sistema OPIS.

Uma das razões para isto é o facto da perfomance do sistema ser negativamente afectada pela rigidez do procedimento de procura. Este impunha uma *perspectiva centrada no processo*, isto é, uma programação dos processos um por um. O procedimento era eficiente no que respeita à *redução dos stocks* mas tinha dificuldade em optimizar a *utilização dos recursos congestionados*.

Alguns aspectos salientes do ISIS são descritos a seguir.

4.1.1. Características Gerais

Usa uma estratégia de procura tipo *best-first em faixa* (ou *beam-search*) num espaço de estados de programas parciais alternativos. Pontos salientes:

- Cada ordem é programada separadamente pela sua prioridade (determinada pela categoria e “due date”).
- Procura para a frente, a partir do tempo de início da ordem, ou para trás, a partir da “due date”.
- Dispõe de operadores para geração de operações, máquinas e tempos de operação alternativos.
- Cada estado é um programa parcial (um caminho até à solução é um programa completo).
- Constrangimentos originados pelo exterior do sistema, ou virem do modelo da fábrica (e usados na procura quando aplicáveis).
- A propagação de constrangimentos advém das decisões feitas em cada estado na procura (que limitam decisões em estados posteriores - não há “lookahead”).

4.1.2. Tipos de Constrangimentos

Há 5 tipos:

- **Constrangimentos de objectivos organizacionais**:
  - “due dates” (os atrasos afectam o cliente)

- níveis de WIP (significam investimento em matérias-primas e valor acrescentado, a minimizar)
- níveis de recurso (de pessoal, de matérias-primas de ferramentas, a manter adequadamente)
- custos (de materiais, vencimentos e de oportunidade perdida, a reduzir)
- níveis de produção (que condiciona, por exemplo, o número de turnos de trabalho em áreas da fábrica)
- estabilidade da fábrica (mínimizar o número de revisões do programa e as perturbações que daí advêm).

- **Físicos** - Características que limitam a funcionalidade (por exemplo das máquinas).
- **Restrições causais** - Condições a satisfazer antes de iniciar uma operação:
  - Precedência - Ordem entre as operações
  - Requerimentos de recurso - Disponíveis antes da, ou durante a, execução de cada operação
- **Disponibilidade** - Durante a construção do programa, recursos afectados a uma operação, poderão não estar disponíveis para outra.
- **Preferência** - Por máquinas, operações, ou sequências de ordens (devida a razões de qualidade, ou de custo).

4.1.3. Programação Reactiva

Uma capacidade limitada para reagir a mudanças na fábrica (falhas de máquinas, falta de materiais e não disponibilidade de ferramentas, baixas de pessoal) que invalidam o programa foi implementada no ISIS (-2), cujo funcionamento (no entanto não testado na fábrica) é de acordo com o seguinte:

- Reservas de actividades invalidadas são canceladas
- Para cada actividade a juzante daquelas e nas ordens respectivas, as reservas de tempos de recursos são canceladas
- Cada uma das reservas canceladas passa a ser um constrangimento de preferência e cada ordem afectada é então, recalendarizada.

4.2. OPIS

Com o projecto do sistema OPIS (Opportunistic Intelligent Scheduler) [Smith 1987], [Smith 1990], [Smith 1994] continuaram-se, a partir de 1985, os esforços iniciados com o sistema ISIS.

4.2.1. Características Gerais

1 *Blackboard* - O sistema OPIS usa uma arquitectura baseada no conceito de *blackboard* que fornece a infra-estrutura para implementação de métodos e estratégias de revisão de programas baseada em constrangimentos (a arquitectura *blackboard*, usada em sistemas periciais, tem a vantagem de permitir a coexistência e cooperação de várias fontes de conhecimento úteis na resolução de um problema, de uma forma modular, no mesmo sistema).

2 Distinção entre *recursos congestionados* e não congestionados e reconhece-se que podem *surgir congestionamentos novos* durante a geração do programa.

3 Sistema *multi-perspectiva*, combina duas perspectivas de Scheduling:

- Perspectiva *baseada em recursos* - Usada em primeiro lugar para programar recursos congestionados.
- Perspectiva *baseada em processos* - Usada a seguir para programar operações não envolvidas com recursos congestionados processo a processo.

4 *Scheduling oportunístico* - Repete a análise inicial para detectar congestionamentos sempre que foi programado um recurso, ou um processo, inteiro. Isto permite detectar o aparecimento de novos congestionamentos durante a construção do programa e rever a estratégia actual de programação se for necessário. Trata-se de um *macro-oportunismo* (é necessário programar um congestionamento completo, antes de focar a atenção noutro) uma forma limitada de oportunismo pois nem sempre os congestionamentos se estendem a todo o horizonte temporal de programação, e podem mesmo sofrer deslocamentos antes de serem inteiramente programados.

5 *Reactividade* - Prevê também programação reactiva, baseada na revisão/adaptação incremental do programa.

4.2.2. Arquitectura

Componentes da arquitectura (baseada num *blackboard*):

Fontes de Conhecimento

- **Métodos de programação** - Executam tarefas de programação; modificam representação da solução actual.
- **Actualização do modelo** - Fonte adicional invocada quando há indicação do exterior de modificações nos constrangimentos (alterações de requerimentos, actualização do status de execução).
- **Análise** - Fornecem informação necessária à formulação de tarefas de programação.

Componentes Adicionais

São a infra-estrutura para programação oportunística e multi-perspectiva:

- **Subsistema de manutenção de programas** - Mantém uma representação dos actuais constrangimentos da solução.
- **Gestor de nível de topo (TLM)** - Coordena o uso dos métodos de programação, análise e actualização do modelo. Contém um ciclo dirigido por eventos onde é aplicado o conhecimento das estratégias de programação baseada em constrangimentos.

4.2.3. Modelação

Modelo de Domínio

Contém a especificação dos constrangimentos e objectivos na execução de processos e afectação de recursos a ser tidos em conta no ambiente alvo. A representação do programa e os métodos para sua análise e modificação são definidos relativamente ao modelo.

Protótipos de Processos

Planos de produção (não instanciados) e são representados por hierarquias de operações. Operações agregadas contêm, ou subprocessos (sequências de operações) mais detalhados, ou conjuntos de alternativas exclusivas. A descrição de uma operação (a qualquer nível da hierarquia) contém:

- Relações de precedência (operações antecessora e sucessora).
- Constrangimentos de duração.
- Requerimentos de capacidade e de preparação.

Recursos

Recursos elementares são agrupados em colecções de recursos maiores para fornecer uma descrição dos constrangimentos de afectação de recursos (capacidade disponível, horas de operação) a cada nível de abstracção nos protótipos de processos.

- *Recursos unitários* - Afectáveis a um só processo (ex.: máquina)
- *Recursos de lote* - Afectáveis a vários processos num mesmo intervalo de tempo (ex.: forno).
- *Recursos conjuntivos e disjuntivos agregados* - Afectáveis a vários processos sem sincronização temporal (ex.: máquina, grupo de operação).

4.2.4. Constrangimentos

Subsistema de Manutenção de Programas

O programa é obtido da instanciação de um protótipo de cada processo a existir no horizonte de programação, cujas operações deverão ser programadas para obter uma solução completa. Os constrangimentos mantidos pelo subsistema de manutenção de programas são dos tipos:

- **Limites temporais** - Tempos de início mais cedo e de fim mais tarde, de cada operação a ser, ou já, programada.

- **Capacidade disponível actual de recursos** - A cada nível da hierarquia ao longo do tempo: sequência de intervalos do tipo (Tinicial,Tfinal,CapacidadeDisponível), cobrindo todo o horizonte de programação.

Actualização dos Constrangimentos

Limites temporais e capacidade disponível de recursos são actualizados como resultado de *decisões de programação* (do sistema), e constrangimentos vindos de *eventos externos*, que são combinados pelo subsistema de manutenção de programas com os constrangimentos já existentes na execução de processos, utilização de recursos e constrangimentos especificados (datas de lançamento e "due dates" de processos).

Propagação de Constrangimentos

Resulta de modificações no programa, podendo originar:

- **Conflitos temporais** - Tempos limite, ou tempos de execução, de operações do mesmo processo violam constrangimentos de precedência, ou limites temporais violam um limite absoluto da execução do processo (data limite).
- **Conflitos de capacidade** - Requerimentos de recurso de operações programadas excedem a capacidade disponível de um recurso durante um intervalo de tempo.

4.2.5. Análise de Conflitos

Métricas para Análise de Conflitos

Servem para resumir as características dos constrangimentos da solução actual e são usadas na análise da estrutura do problema. Limitam-se a um *horizonte* (temporal) *de conflito*: região localizada da solução contendo (também) as operações directamente envolvidas no conflito.

Métricas para Estimar a Severidade do Conflito:

- **Duração do conflito** - Grandeza temporal da inconsistência, normalizada em relação à média da duração de todas as operações programadas no recurso em questão, dentro do horizonte de conflito. Dá uma medida da validade continuada do programa do recurso: baixa duração indica que as decisões de sequenciação já feitas sobre o recurso permanecem válidas (problema reside em detalhes de "timing").
- **Tamanho do conflito** - Número mínimo de operações envolvidas no conflito (abrangidas pela duração) que devem ser retiradas e recalendarizadas para repor a consistência, exeptuando a operação que o desencadeou (que não é retirada). Baixo tamanho de conflito indica que as decisões de sequenciação já feitas sobre o recurso continuam válidas (senão, optimização da sequência no recurso é importante).

Métricas para Caracterizar a Folga dos Constrangimentos:

- **Tempo de recurso parado** - Média da quantidade de capacidade disponível no recurso em questão, dentro do horizonte de conflito. Baixo tempo de recurso parado indica situação de congestionamento, em que a optimização do programa do recurso é importante.
- **Folga local a montante** - Média das folgas a montante de todos os processos programados no recurso em questão, dentro do horizonte de conflito. Folga a montante de um processo é a diferença entre o tempo de início programado para a sua operação no recurso e o tempo

de fim programado (ou real) da operação precedente do mesmo processo. Mede flexibilidade dos tempos de início das operações programadas num recurso: com folga local a montante alta há oportunidades de resequenciação no recurso.

- **Folga local a juzante** - Média das durações do primeiro atraso (programado) nos programas a juzante de cada processo programado no recurso em questão, dentro do horizonte de conflito. Mede a flexibilidade dos tempos de fim das operações programadas num recurso. Como num programa de boa qualidade só tem tempos de espera antes de recursos congestionados, se a folga a juzante é alta isso indica que há, pelo menos, um congestionamento a juzante (e aí, a optimização dos programas de recurso pode ser importante); se é baixa não há, aparentemente, congestionamentos a juzante..
- **Atraso projectado** - É definido relativamente a operações em conflito num recurso e é a diferença entre o tempo de chegada mais cedo do processo ao congestionamento a juzante, e o tempo de início programado para o mesmo processo no recurso em questão. Se não há congestionamento então é tomada a diferença entre o tempo mais cedo de fim do processo e a sua "due date". Mede uma flexibilidade temporal relativa às operações num conflito: se o atraso projectado é positivo a optimização de recurso é importante.
- **Variância do atraso projectado** - É definida relativamente a todas as operações programadas no recurso em questão, dentro do horizonte de conflito. Se é alta então pode ser possível balancear atrasos projectados negativos e positivos de determinados processos, trocando as procuras entre pares (de programas) deles.

4.2.6. Reparação (Reactiva) de Programas Baseada em Constrangimentos

Métodos básicos para revisão de programas aplicados na resposta a problemas reactivos de programação (resposta a conflitos) mediante o valor das métricas de análise de conflitos:

Calendarizador de Ordens (OSC)

Revê o programa de uma sequência contígua de operações no plano de um dado processo (ex.: associado a uma determinada ordem de produção). Retira as atribuições de tempo e recursos de cada operação, aplica uma estratégia tipo "beam-search" do ISIS melhorada (com uma estratégia também baseada na decomposição de recursos e não só na base do processo) e determina então as novas atribuições para a sequência de operações. Pode funcionar segundo 2 *níveis de visibilidade*:

- **CV-OSC (Visibilidade completa)**: a procura considera só os intervalos de execução com capacidade de recurso disponível actualmente.
- **PV-OSC (Visibilidade prioritizada)**: a procura considera também a capacidade afectada a processos com prioridade mais baixa. Pode introduzir *novos conflitos de capacidade* no programa.

Calendarizador de Recursos (RSC)

Resequencia operações num determinado recurso (ou grupo de recurso substituível) a partir de um ponto do tempo especificado para acomodar consistentemente um dado conjunto de operações em conflito (isto pode demorar mais de 1 ciclo). Pode introduzir *novos conflitos temporais* no programa actual em operações a juzante no processo.

Deslocador à Direita (RSH)

Desloca os tempos de execução das operações designadas para a frente no tempo (saltando por cima de operações programadas no mesmo recurso). *Não introduz conflitos* porque, embora possam surgir novos conflitos temporais (com operações programadas a juzante no mesmo processo) e/ou novos conflitos de capacidade (com operações programadas a juzante no mesmo recurso) eles são resolvidos internamente com a necessária propagação recursiva dos deslocamentos através dos programas de processo e recurso.

Deslocador à Esquerda (LSH)

Método reactivo semelhante, *não disruptivo*, que desloca os tempos de execução das operações para trás no tempo, na medida em que a disponibilidade de recursos e os constrangimentos temporais de processos o permitam (é usado na actual implementação apenas para responder a eventos de oportunidade).

Alternador de Procuras (DSW)

Troca pares de afectações de recursos e intervalos de execução em dois processos similares (aplicável em situações em que um processo está atrasado, se houver um processo com o qual seja possível fazer a troca).

4.2.7. Extensões e Estado Actual

Em [Smith 1994] referem-se linhas de investigação no scheduling da produção que exploram variantes da metodologia usada no OPIS (para as referências ver [Smith 1994]):

Smith & Hynynen (1987) e Hynynen (1988)

Gestão de programas do OPIS é estendida numa base de *decomposição estrutural da fábrica*, com hierarquias dos constrangimentos de tempo e capacidade de recurso.

Ow, Smith & Howie (1988)

No sistema CSS a responsabilidade de programação é distribuída por um conjunto de *agentes negociadores de recursos* (cada um preocupado com a utilização eficiente de um conjunto de recursos) e um *agente gestor de ordens* (preocupado com objectivos e constrangimentos relacionados com processos).

Smith, Keng & Kempf (1992)

Responsabilidade dividida entre um *calendarizador global* e um conjunto de *agentes de "despacho" locais*, para tirar partido de flexibilidade temporal no programa global para reparação local de tempo de execução do programa (para controlo em tempo real de fábrica de "wafers" da INTEL).

Smith & Sycara (1993)

O sistema DITOPS aplica a metodologia de reparação de programas básica do OPIS ao problema da gestão distribuída de programas de transporte em larga escala (no âmbito de crise militar).

Muscettola et al. (1992)

O sistema HSTS tem uma arquitectura que integra planeamento e programação. Explora a revisão de programas baseada em constrangimentos usando as representações mais flexíveis

de uma “base de dados temporal” dos constrangimentos da solução, em que tempos de início e de fim não são fixos, mas são confinados a intervalos que satisfazem os constrangimentos temporais existentes.

4.3. SONIA

O sistema SONIA [Le Pape 1988a], [Le Pape 1994] é também baseado no conceito de *blackboard*. Foi aplicado a problemas de programação predictiva e reactiva do tipo job-shop não-preemptivo envolvendo datas de lançamento e datas limite, precedências de operações, constrangimentos de duração, tempos de movimentação e preparação, especificações de turno e constrangimentos disjuntivos de recursos partilhados.

4.3.1. Características Gerais

- *Modelação* - Representação de entidades do domínio (programas, hierarquias de recursos e de processos, tipos de constrangimentos) feita de modo semelhante à do OPIS.
- *Blackboard* - Fontes de conhecimento tomam parte nas actividades de programação (predictivas, reactivas e de análise) e são enquadradas numa arquitectura do tipo *blackboard*. Esta permite integrar, no mesmo sistema, algoritmos matemáticos, conhecimento do perito em programação, conhecimento dependente da fábrica e capacidades interactivas.
- *Scheduling macro-oportunístico* - Cada fonte executa uma série de decisões (mais de uma) de cada vez que é aplicada (quer dizer põe-se aqui ao sistema, também o problema do controlo/adaptação do comportamento de fontes de conhecimento de programação para a situação específica).
- Componentes de programação *predictiva* e *reactiva*.
- *Propagação de constrangimentos* - Ênfase no controlo inteligente da propagação de constrangimentos.
- *Heurísticas* - As fontes de conhecimento podem usar heurísticas diferentes conforme a situação em cada momento. É dado especial ênfase à selecção das heurísticas a empregar.

4.3.2. Arquitectura

Construído sobre uma arquitectura tipo *blackboard* com os seguintes componentes:

Fontes de Conhecimento Predictivo e Reactivo

Servem para construir e modificar programas:

- **Selecção global** - Serve para escolher (mediante heurísticas) um conjunto de operações (a executar dentro de turnos de trabalho abertos), que requerem recursos em sub-carga, para afectação de recursos. A selecção de uma operação altera-lhe o *status de programa* (para *seleccionado*) e os constrangimentos relevantes são criados e propagados pelo sistema de gestão de programas.
- **Ordenação** - Usado para realizar decisões de ordenação (mediante heurísticas) quando há constrangimentos disjuntivos (p.ex.: partilha de recursos) a satisfazer. É um processo iterativo de satisfação de constrangimentos. Quando falha (p.ex.: quando a *fonte de selecção* seleccionou operações em demasia) algumas das menos importantes são rejeitadas pela *fonte de rejeição*.
- **Selecção de ordem** (reactivo) - Para seleccionar operações do mesmo plano de produção. Pára quando a uma operação não pode ser seleccionada dentro do turno de trabalho aberto.

Usada em geral, quando a *fonte de rejeição* rejeitou operações, e permite melhoria do programa.

- **Rejeição** (reactivo) - Rejeita operações seleccionadas (escolhidas mediante heurísticas) e é empregue quando a *fonte de ordenação* falha ou quando eventos inesperados impedem a execução do programa predictivo. A rejeição de uma operação altera-lhe o *status de programa* (para *ignorado*) e faz com que os constrangimentos e conflitos envolvendo a operação sejam removidos pelo sistema de gestão de programas.
- **Recalendarização global** (reactivo) - Usado para processar e modificar o plano a partir da data actual. Dispõe de 2 estratégias: (1) Deslocamento à direita das operações em cada recurso permanecendo elas na mesma ordem em que estavam (2) Permutações de modo a optimizar o plano e reduzir os efeitos do atraso - usado quando há tempo para corrigir o plano.

Fontes de Conhecimento de Análise

Avaliam contextos predictivos e reactivos de resolução de problemas.

- **Análise de capacidade** - Detecta recursos congestionados ou em sub-carga. Funciona como no OPIS (constrói temporàriamente um programa predictivo grosseiro, calculando a capacidade e a procura a partir das reservas existentes, para certos recursos).
- **Análise de conflitos** - Examina em detalhe um conjunto de conflitos para propor depois as reacções apropriadas.

Fontes de Conhecimento de Interface -

- **Interface do utilizador** - Permite que o utilizador construa e modifique programas (seleccionar e rejeitar, ordenar e permutar, interromper operações, afectar um recurso a uma operação durante um intervalo de tempo, adicionar e remover turnos de trabalho, actualizar datas de lançamento e limite, adicionar e remover constrangimentos na linguagem de constrangimentos disponível e activar o sistema de propagação de constrangimentos e visualizar os resultados num gráfico de Gantt).
- **Aplicação de Programa** - Fornece ao utilizador o programa gerado ou corrigido (invocada sempre que um programa novo é considerado aceitável).
- **Controlador de execução** - Informa o sistema de gestão de programa do curso dos eventos na fábrica.

Fontes de Conhecimento de Controlo

De acordo com os resultados da análise escolhem as fontes de conhecimento a aplicar e que heurísticas elas devem usar (ajustando assim o seu comportamento ao contexto). Sempre que uma fonte é invocada gera eventos que são colocados ou no *blackboard de domínio* (eventos relacionados com o estado actual e progresso do processo de tomada de decisão de programação) ou no *blackboard de controlo* (eventos relacionados com o processo de controlo) ou em ambos. Para além das fontes normais (inerentes ao controlo num sistema de *blackboard*) existem mais as seguintes:

- **Controlador de heurísticas** - Activa ou desactiva heurísticas do conjunto de heurísticas usadas pelas fontes de conhecimento de programação, mediante informação obtida através das fontes de conhecimento de análise de capacidade.

- **Controlador da propagação de constrangimentos** - Pode modificar o conjunto de regras de controlo usadas na propagação de constrangimentos antes da activação de uma fonte de conhecimento, de modo a regular a quantidade de propagação que ocorre após uma tomada de decisão de programação.

Sistema de Gestão de Programas

Construído sobre o sistema de propagação de constrangimentos, recebe informação de *decisões de programação* (de acordo com o status de operações, datas de lançamento e limite, capacidade dos recursos) e de *eventos inesperados* (falhas de máquinas, atrasos, na fábrica) e mantém o *programa*, criando constrangimentos temporais e de reserva, que envia para o sistema de propagação. Este último deriva novos constrangimentos e detecta inconsistências, fornecendo descrições de *conflitos*.

4.3.3. Controlo do Comportamento das Fontes de Conhecimento

O comportamento das fontes "macro" de conhecimento tem de ser controlado do (seu) exterior, por isso uma fonte tem os seguintes componentes:

- **Condição** - Indica se a fonte deve estar activa.
- **Acção** - Indica os eventos que a fonte produzirá ao ser activada. É um procedimento parcial.
- **Comportamento** - Pontos de flexibilidade, a estabelecer dinâmicamente (para completar o procedimento parcial da fonte) de acordo com o contexto. Trata-se do controlo de: *heurísticas* (a activar/desactivar), *propagação de constrangimentos* (regras de controlo de propagação usadas para regular a quantidade de propagação de constrangimentos), *estratégias de backtracking* (cronológico/selectivo) e *consideração de critérios de qualidade* (preferências/restrições).

4.3.4. A Propagação de Constrangimentos

Em SONIA é dado um especial ênfase aos constrangimentos e ao controlo inteligente da sua propagação.

Constrangimentos

Especificam valores cuja atribuição é admissível a variáveis (p.ex.: constrangimentos de domínio descrevem domínios de valores possíveis - *unários*; relações entre variáveis definem um subconjunto do produto cartesiano entre esses domínios - *binários*, ou *n-ários*).

Representação de Variáveis e de Constrangimentos

Usam-se *hipergrafos* em cujos *vértices* representam as *variáveis* (com os domínios associados) e cujos *hiperarcos* representam as *relações entre variáveis*. No caso de *constrangimentos binários* (entre 2 variáveis) usa-se um *grafo*.

Propagação de Constrangimentos

Entendida como uma actividade dedutiva, executada por um sistema de propagação de constrangimentos, em que são derivados constrangimentos novos a partir dos já existentes e detectadas inconsistências entre constrangimentos. Esta actividade permite *decompor o problema sem deixar de tomar em conta as interacções entre os subproblemas*.

Quanto aos tipos de passos na propagação de constrangimentos ([Dechter 1989]) podemos ter:

- **Abordagem combinatória** - Executam-se operações no que respeita aos valores das variáveis. Usadas em geral em domínios finitos ([Kumar 1992]).
- **Abordagem algébrica** - Executam-se operações no que respeita a relações entre os valores (desconhecidos) das variáveis. Usadas em domínios infinitos com estrutura homogénea e para raciocinar sobre tempo ([Allen 1983], [Ladkin 1988]).

Conflitos

Quando há constrangimentos não compatíveis, o sistema de propagação de constrangimentos deve fornecer ao sistema que o utiliza uma descrição dos conflitos, de modo a este poder retroceder (*backtracking*) sem recriar as mesmas condições, sendo necessário que memorize que constrangimentos foram originados de que outros. Usam-se então técnicas semelhantes às dos sistemas de ATMS (Assumption-based Truth Maintenace Systems).

Sistemas Incompletos

Surgem problemas de complexidade quando há disjunções (determinar se um conjunto de constrangimentos envolvendo constrangimentos disjuntivos é consistente é NP-difícil). Por isso faz-se um compromisso entre a detecção de contradições e o tempo computacional dispendido na propagação de constrangimentos (o algoritmo usado não detectará certas interacções entre constrangimentos).

Há 2 tipos de sistema incompletos:

- **Propagação com limite fixo** - Executam quantidade de propagação de constrangimentos fixa à partida.
- **Propagação com limite variável** - Sistemas flexíveis em que a propagação é reduzida ou aumentada com baso num conjunto prè-determinado de certos parâmetros (p.ex.: intervalos de referência, ver [Allen 1983]), ou incluem uma pequena linguagem de programação para controlo da propagação ([Le Pape 1988b], [Van Hentenrick 1989]).

Quantidade Óptima de Propagação

A experiência com o último tipo de sistemas de propagação de constrangimentos indica que a máxima eficiência na resolução de problemas varia com o *sistema* de resolução de problemas, a *aplicação* (de fábrica para fábrica dentro do mesmo tipo - job shop, etc.) e o *contexto* (p.ex.: no caso de urgência, limita-se a propagação a constrangimentos sobre operações cuja execução está próxima)

Sistema de Propagação de Constrangimentos

Sistema de inferência controlável, com:

- **Regras de dedução** - Indicam que inferências poderão fazer-se a partir de um conjunto de premissas.
- **Regras de controlo** - Especificam que inferências devem ser feitas em cada ponto do processo de resolução do problema.

Subsistemas Especializados em SONIA

Fazem parte do sistema de propagação de constrangimentos de SONIA

- Lógica proposicional.
- Relações de ordem em conjuntos infinitos totalmente ordenados (aceitando igualmente, fórmulas disjuntivas, conjuntivas e negativas).
- Relações entre intervalos tipo Allen ([Allen 1983]) e disjunções de intervalos (aceitando igualmente, fórmulas disjuntivas, conjuntivas e negativas de relações de intervalos).
- Constrangimentos de duração - distâncias mínimas, máximas e conhecidas entre pontos temporais (aceitando igualmente, fórmulas disjuntivas, conjuntivas e negativas de constrangimentos de duração).
- Constrangimentos de reservas (que servem para a propagação manter horários e detectar conflitos de capacidade).
- Famílias várias de constrangimentos de domínio, incluindo igualdades e desigualdades (quando o domínio global, definido pelo utilizador, é totalmente ordenado).

4.3.5. Conclusões

SONIA gera programas correctos, reage quando um evento inesperado impede a execução do programa e atinge o equilíbrio quando dispõe de um programa actualizado correcto.

Não é *completo em termos de optimização* porque não pode ser usado para minimzar, ou maximizar, o valor de uma função de custo, ou de avaliação. No entanto, o uso de heurísticas apropriadas e o facto de poder aplicar algumas das fontes de conhecimentos para melhorar um dado programa (a fonte de análise de capacidade pode detectar sub-cargas e a de selecção global repará-las) permitem a geração e manutenção de *soluções satisfatórias* para o problema de scheduling.

A comparação da perfomance de SONIA com a de uma regras de prioridade orientadas para a data limite mostrou que o SONIA é melhor em 5% para o número de operações de preparação, 10% para custos de preparação, 5% para o tempo de máquinas paradas, 10% (em certos casos mais de 20%) para o número de ordens em atraso e -5% para o atraso relativo (tardiness) médio (a regra de prioridade é aqui melhor por ser muito mais especializada neste aspecto).

O autor refere como maior dificuldade na construção destes sistemas a *identificação do conhecimento de controlo* e a sua *expressão* na linguagem de controlo disponível.

Temas de interesse para o futuro referidos pelo autor são:

- Tornar a adaptação de um sistema controlável (como o SONIA) gerível pelos utilizadores.
- Ter um sistema com capacidade de *adquirir conhecimento de controlo* a partir da sua experiência e adaptar-se ao tipo de problema que encontra.

4.4. Micro-Boss

O Micro-Boss (Micro-Bottleneck Scheduling System) [Sadeh 1991], [Sadeh 1994] é um sistema de apoio à decisão para Scheduling da produção em fábricas (desenvolvido na Universidade de Carnegie Mellon) orientado para gerar e manter programas de produção de alta qualidade.

4.4.1. Características Gerais

- Ênfase na *análise do espaço de soluções* realizada na procura onde se empregam técnicas de *look-ahead* para medir a competição pelos recursos e heurísticas para identificar e programar operações críticas.
- *Scheduling micro-oportunístico* - Monitora constantemente a competição pelos recursos, durante a geração de um programa, sendo o esforço para resolução do problema continuamente direccionado para o recurso mais congestionado. Isto resulta numa perspectiva *baseada em operações* (cada operação é considerada um ponto de decisão independente, podendo ser programada sem ser necessário que outras operações usando o mesmo recurso, ou pertencendo ao mesmo processo, o sejam).
- Integra componentes de programação *predictiva*, *reactiva* e *interactiva*.
- Entende como factores críticos prioritários na qualidade do programa a satisfação das *datas limite* e a minimização dos *custos de posse*.
- Provou-se experimentalmente que as heurísticas de Scheduling micro-oportunísticas produzem frequentemente melhores resultados que outras técnicas menos flexíveis centradas em congestionamentos (OPT, OPIS), em especial em problemas com operações com *janelas temporais rígidas*, na *reparação de programas* e na *integração em sistemas interactivos* em que se podem intercalar decisões de Scheduling automáticas e manuais.

4.4.2. O Procedimento de Procura Micro-Oportunístico

Objectivo de Programação

É, para além de satisfazer os constrangimentos, minimizar os *custos de atraso* relativamente às datas limite e os *custos de posse*. Os custos associados a cada processo j são:

- **Custo de atraso marginal** ($tard_j$) - É o custo por cada unidade de tempo de atraso do processo (relativamente à sua data limite) que, multiplicado pelo número de unidades de tempo do atraso, dá o custo do processo $TARD_j$.
- **Custo marginal de posse** (stocks de produtos em curso e acabados) (inv^j_i) - Cada operação O^j_i introduz, incrementalmente, um custo por unidade de tempo, não negativo de stock (inv^j_i); estes custos somam-se para obter o custo do processo INV_j.

O custo total é dado por: $\sum_{j=1}^{n} (TARD_j + INV_j)$.

Micro-oportunismo

É uma abordagem flexível à programação. Pára de programar operações num recurso (congestionado) assim que outro se revele mais congestionado. Existindo vários congestionamentos isto permite ao sistema desviar a atenção de um congestionamento para

outro durante a geração do programa (em vez de se concentrar na optimização de um só congestionamento à custa dos outros).

Algoritmo de Procura Micro-oportunística

Dado um problema de programação, o Micro-Boss começa num estado de procura em que nenhuma operção está programada (ou alternativamente num estado com um programa parcial que se pretende completar) e segue os seguintes passos:

1- Parar se todas as operações estiverem programadas.
2- Aplicar o procedimento de *imposição de consistência*.
3- Se se detectou um estado sem saída executar *backtracking* (cronológico).
4- Se as houver, programar as operações (não programadas) que ficaram com 1 só reserva possível (criando um novo estado de procura para cada uma).
5- Aplicar o procedimento de *análise look-ahead*.
6- Seleccionar a próxima operação a ser programada (heurística de ordenação das operações).
7- Seleccionar uma reserva para essa operação (heurística de ordenação das reservas).
8- Criar um novo estado de procura juntando a nova atribuição de reserva ao programa parcial actual. Continuar em **1-**.

Imposição de Consistência

Procedimento que actualiza o conjunto de reservas possíveis para cada operação ainda não programada (podando o espaço de procura ao reduzir a possibilidade de atingir um estado sem saída e limitar o trabalho dispendido na exploração de alternativas sem futuro). Cada operação não programada tem um par de tempos de início mais cedo/mais tarde e são assinaladas como não disponíveis as reservas afectadas a operações já programadas.

Análise Look-ahead

Executado em cada estado de procura, serve para ajudar o sistema a focar o esforço nos conflitos que, em cada estado, parecem ser mais críticos (com maior impacto na qualidade de todo o programa) para produzir programas de melhor qualidade (focando o esforço primeiro nestes conflitos, o sistema tem o maior número de opções quanto possível para os optimizar, pois, assim que os compromissos críticos foram resolvidos as operações que restam para programar tendem a ser mais desacopladas, e mais fáceis de optimizar) e limitar a quantidade de *backtracking* (à medida que a competição de recursos diminui, diminui também a probabilidade de *backtracking*).

Passos do procedimento de análise look-ahead:

1. **Optimização das atribuições de reservas** dentro de um processo - As reservas possíveis de cada operação não programada são ordenadas de acordo com quanto minimizam os custos do processo a que pertencem (heurística da ordenação das reservas). Para medir a competição por recursos, para cada tempo de início τ de cada operação (não programada) O^j_i é implìcitamente mantida uma função $mincost^j_i(\tau)$ indicando o mínimo custo adicional no processo j se O^j_i começasse no tempo τ em vez de num dos seus melhores tempos de início (por definição, se τ for um desses tempos então $mincost^j_i(\tau) = 0$). Cada um dos valores desta função advém de 2 parcelas, relativas ao custos aparentes de atraso marginal e de posse.
2. **Avaliação da competição pelos recursos** ao longo do tempo - Recursos/intervalos de tempo onde a competição é maior ajudam a identificar a operação crítica, a ser programada

a seguir (heurística da ordenação das operações). A importância de um conflito depende do número de operações a competir pelo mesmo recurso, a extensão do intervalo de tempo de sobreposição entre os requerimentos destas operações, o número de reservas alternativas ainda disponíveis para cada uma destas operações em conflito e os seus custos (determinados pelas funções mincost).

A cada tempo de início possível para cada operação não programada é associada uma *probabilidade subjectiva* $\sigma^j_i(\tau)$ de que cada possível tempo de início τ de uma operação O^j_i (não programada) seja para ela seleccionado no programa final (tempos de início possíveis com custo mincost mais baixos têm probabilidade mais alta, para reflectir a expectativa de que produzirão melhores programas). Com base nela, é calculada a *procura individual* D(t) da operação O^j_i pelo recurso R^j_i , que é a probabilidade de que ela use o recurso no instante de tempo t (sendo ao mesmo tempo uma medida subjectiva da expectativa da operação na disponibilidade do recurso no tempo t). O *perfil de procura agregado* das operações não programadas por um recurso é obtido adicionando a procura individual de cada uma dessas operações por esse recurso e é uma medida da competição delas por esse recurso.

Selecção de Operação

As operações críticas são aquelas cujas boas reservas (identificadas no 1º passo da análise para a frente) colidem com as boas reservas de outras. O pico maior nos perfis de procura agregados determina o próximo conflito a ser optimizado, pois permite identificar a operação com a maior contribuição individual para esse pico (a que tem maior expectativa na disponibilidade da reserva - recurso/intervalo de tempo). As boas reservas possíveis para essa operação são as que têm maior probabilidade de ficar não disponíveis se as outras operações competindo no mesmo micro-congestionamento fossem programadas primeiro.

Selecção de Reserva

Para programar a operação crítica determinada na selecção de operação identifica-se uma reserva, para essa operação, que reduza o mais possível os custos do processo a que a operação pertence e dos outros processos com os quais a operação compete. Usa-se uma variante do procedimento do problema de programação do tipo de 1 máquina avanço/atraso, em que o programa de uma máquina com menor custo é usado para determinar a reserva a atribuir à operaçõa crítica.

4.4.3. Programação Reactiva e Interactiva

O Micro-Boss dispõe capacidades de programação reactiva para atender a factores como a duração variada das operações, falhas de máquinas, atraso na chegada de matérias-primas, chegada de novas encomendas, cancelamento de encomendas existentes, etc.. Embora as pequenas disrupções (por exemplo, pequenos desvios na duração das operações) não impliquem necessàriamente modificação do programa, já não será o mesmo com o seu efeito acumulado.

Níveis de Tratamento das Disrupções

Dependendo da gravidade e tempo de resposta necessário:

- **Nível de controlo** - São tratadas as pequenas disrupções que necessitam de respostas rápidas (baseadas numa visão muito restrita/local do problema) por meio de heurísticas de

controlo (por exemplo, processar primeiro a operação com a data de início programada mais cedo, ou quando uma máquina falhou, reconduzir os processos críticos para máquinas equivalentes, se as houver).

- **Nível de programação** - Para desvios mais graves do programa o nível de controlo invoca o módulo de programação do Micro-Boss para reparar/reoptimizar o programa a uma perspectiva mais global, (enquanto se vai atendendo a decisões mais imediatas). Isto produz uma melhor reparação do programa mas demora mais tempo que a acção do nível de controlo, por se basear em considerações mais globais.

Há um compromisso entre a rapidez de resposta do sistema e a quantidade de reoptimização realizada:

- **Ambientes de produção** - As disrupções são frequentes, grande parte serão tratadas ao nível de controlo.
- **Ambientes menos caóticos** - Maior proporção de disrupções tratada ao nível da programação.

Reparação de Programas

O Micro-Boss tenta ter uma visão mais global do problema e aproveitar as capacidades dos procedimentos de procura micro-oportunísticos, com a reparação de programas em 2 fases:

1. Identificação das operações a serem reprogramadas e desprogramação das mesmas.
2. Resolução do problema de programação composto por aquelas operações e os constrangimentos impostos nelas pelas já executadas e as não desprogramadas, pelo módulo micro-oportunístico de programação.

Em geral, o problema da fase 2- tem solução, mas se isso não acontece é necessário voltar à fase 1- e desprogramar um maior número de operções. Isto pode ser evitado de antemão desprogramando-se, logo na fase 1-, mais operações do que o aparentemente necessário (o que até pode ser vantajoso porque, apesar do espaço de procura resultante ser maior, poderá conter melhores soluções de reparação).

Programação Interactiva

O Micro-Boss dispõe capacidades de programação interactiva para manipulação do programa para atender a constrangimentos ad hoc e preferências que ocorrem ocasionalmente, ou que variam ao longo do tempo, apoiar o utilizador na identificação de fontes de ineficiência (ordens atrasadas, recurso sobrecarregados, etc.) e opções possíveis para as corrigir (adicionar mais tempo/turnos nos recurso, reconduzir ordens por outro percurso, etc.), na exploração de cenários e alternativas possíveis (por exemplo, decidir se se devem deixar acabar processos atrasados excedendo datas limite ou introduzir trabalho/turnos suplementares).

Permite ao utilizador final executar decisões de programação (micro-oportunísticas) intercalando modos manual e automático (sob a supervisão do módulo de imposição de consistência) e analisar, editar, armazenar e comparar (usando métricas variadas como o custo total do programa, atraso ponderado médio, avanço ponderado médio, trabalhos em curso e trabalhos no sistema) programas completos e parciais (usam-se estimativas optimistas no caso de programas parciais em que as métricas não podem ser calculadas de forma exacta) através de uma interface em formato de gráfico de Gantt.

4.4.4. Evolução

O autor refere várias aplicações das heurísticas micro-oportunísticas do Micro-Boss a outros problemas de programação da produção e à programação dos transportes, entre os quais:

- Programação da laminagem no contexto da *produção de aço* de um fabricante Finlandês.
- Programação de *transportes militares* e planeamento de *distribuição* de munições, numa shell (KBLPS - Knowledge Based Logistics Planning) desenvolvida por Carnegie Group, Inc. e LB & M Associates

A investigação actual vai no sentido de aplicar e ampliar a abordagem actual para resolver problemas de programação de produção e transporte.

4.5. HSTS

HSTS (Heuristic Scheduling Testbed System) [Muscettola 1994a] é um quadro geral de representação e resolução de problemas que tenta integrar o Planeamento e a Programação, em que foram desenvolvidos e experimentalmente testados sistemas de planeamento/programação para domínios não convencionais. Um destes domínios é o da programação a curto prazo para o HST - Hubble Space Telescope (produção de comandos detalhados para o telescópio).

4.5.1. Características Gerais

- Integração de *planeamento e programação* - O problema é entendido como um conjunto de restrições e preferências nos valores de um vector de estado, cuja satisfação identifica um plano/programa no conjunto de todos os *comportamentos possíveis*. Funções de avaliação impõem preferências nos comportamentos possíveis do sistema. Tenta-se obter comportamentos com um alto (globalmente máximo, se possível) nível de satisfação das preferências.
- Os constrangimentos têm a ver tanto com a execução de *acções* (como na programação clássica), como com *estados* estacionários (como no planeamento clássico).
- Possibilidade de ter *modelos de domínio* com grande detalhe.
- *Flexibilidade temporal* - Os programas em HSTS identificam um conjunto de comportamentos legais do sistema. Consequentemente há uma menor necessidade de relaxação de constrangimentos (ver procedimento heurístico CPS).
- *CPS* (Conflict Partition Scheduling) - Metodologia de programação heurística usada por HSTS, que opera numa rede de constrangimentos temporais flexíveis com a orientação de estimativas estatísticas das propriedades da rede.

4.5.2. Arquitectura

O HSTS é composto por dois componentes nucleares: HSTS-DDL e HSTS-TDB.

HSTS-DDL (HSTS-Domain Description Language)

Incorpora uma linguagem de descrição do domínio que permite a especificação da estrutura estática e dinâmica de um sistema, dando suporte à expressão de um modelo como um *conjunto modular de padrões de constrangimentos* a satisfazer por qualquer comportamento legal do sistema.

Um *modelo de sistema* (em HSTS-DDL) está organizado como um conjunto de *componentes de sistema*, cada um associado a um *conjunto de propriedades* sendo cada uma, uma entrada no vector de estado, que pode assumir um só *valor* em cada instante. Há *propriedades estáticas*, cujo valor é fixo no tempo (tìpicamente parâmetros do sistema) e *propriedades dinâmicas*, designadas por *variáveis de estado*, cujo valor é variável no tempo (determinam o comportamento do sistema).

A especificação de um modelo de sistema em HSTS pode ser decomposta em vários *níveis de abstracção* em que os componentes de sistema e variáveis de estado em níveis abstractos agregam os de níveis mais detalhados. As relações entre níveis são estabelecidas por descritores de refinamento que mapeiam alguns valores abstractos para uma rede de valores

associada ao nível de maior detalhe seguinte (incluindo correspondência entre tempos de início e fim).

HSTS-TDB (HSTS-Temporal Database)

Dá suporte à construção dos comportamentos legais de um sistema através de princípios de representação de um *time map* ([Dean 1987], [Schrag 1992]) com extensões adequadas aos modelos de HSTS-DDL.

O *token* é a primitiva de descrição temporal. Consiste num intervalo de tempo, identificado pelo *tempo de início* e *tempo de fim*, no qual uma determinada condição é verdadeira. Representa um único segmento de evolução de uma única variável de estado.

A *rede de tokens* é o conjunto de *tokens* e constrangimentos entre eles na base de dados. É estabelecendo constrangimentos temporais e de tipo entre pares de *tokens* que se asseguram as condições necessárias na resolução de um problema. Estas restrições impostas nos *tokens* advêm do próprio problema (por exemplo, uma data de lançamento na ocorrência de uma actividade) e do modelo de sistema em HSTS-DDL (por exemplo, especificações de compatibilidade que requerem a ocorrência de um padrão relacionado de actividades de suporte). A rede de *tokens* é subdividida em *níveis* comunicantes, correspondentes, no HSTS-DDL, a *níveis de abstracção* do modelo de sistema. Existem primitivas para *criar* e *inserir tokens* e criar instâncias de *relações temporais*, bem como para removê-los.

Constrangimentos temporais são organizados num grafo denominado *rede de pontos temporais*, que contém em cada nó um tempo de início, ou de fim de um *token*. Os arcos entre pontos temporais são distâncias temporais derivadas das relações temporais colocadas na base de dados.

A *time line* é uma sequência linear de *tokens* que cobre completamente o horizonte de programação para uma variável de estado (resulta de uma generalização dos perfis de capacidade de recurso usados na programação clássica). Num plano/programa completamente especificado a *time line* é uma sequência de tokens com valores fixos.

4.5.3. Programação do Telescópio

Devido às dimensões do problema e à variedade de interacções entre os constrangimentos a resolução do problema é feita em dois estágios, em dois níveis de abstracção no modelo de domínio de HST que contêm um ciclo de tomada de decisão idêntico:

- **Nível abstracto** - São geradas as sequências de observações tomando em conta a disponibilidade e o tempo de reconfiguração global do telescópio e as janelas de visibilidade dos alvos (o modelo contém uma variável de estado para a visibilidade de cada alvo de interesse e uma variável de estado para representar a disponibilidade do telescópio).
- **Nível detalhado** - Refina-se a solução intermédia de modo a tomar em conta todos os constrangimentos do domínio. São gerados planos/programas que são directamente traduzíveis para comandos do telescópio. Decisões abstractas são expandidas e adaptadas a um modelo de domínio com uma variável de estado para cada subsistema do telescópio. Inicialmente a base de dados temporal contém os programas de observação candidatos no nível abstracto e uma *time line* para cada variável de estado. Cada programa é uma rede de *tokens*, com um *token* de valor para cada pedido de observação, sem que nenhum *token* esteja inserido numa *time line* de variável de estado.

4.5.4. Flexibilidade Temporal na Programação

Em HSTS optou-se pela solução, mais flexível, de estabelecer constrangimentos de sequência entre *tokens* que requerem o mesmo recurso (exemplo, τ_i *before* $([0,+\infty])\tau_j$), como alternativa à solução adoptada na programação clássica de associar valores (recursos e tempos de início e fim) exactos às variáveis (actividades).

Ao raciocinar sobre conjuntos de possíveis atribuições de tempos de início e fim para as actividades ainda não programadas a abordagem temporal flexível tem vantagens sobre a atribuição de valores exactos, pois quando se estabelece um constrangimento apenas se restringe o intervalo de valores possíveis das variáveis, sem limitar necessàriamente a dimensionalidade do espaço de procura. Isto permite manter um grande número de valores possíveis para as variáveis e diminuir assim o risco de o sistema de programação se perder na procura.

Exemplo: Para 2 actividades de duração unitária que requerem o mesmo recurso de capacidade simples e com tempos limite idênticos, com n possíveis tempos de início.

- Sem considerar a limitação de capacidade do recurso há n^2 tempos de início possíveis para o par de actividades.
- Se se fixa o tempo de início de uma actividade restam (n - 1) tempos de início possíveis para a outra que não violam o constrangimento de recurso. O tamanho do espaço de procura é O(n). Alternativamente, se se introduz um constrangimento de precedência entre as duas actividades o número total de valores possíveis para os tempos de início consistentes é ((n - 1) n)/ 2. O tamanho do espaço de procura é $O(n^2)$.

CPS - Heurística de Programação por Partição de Conflitos

CPS [Muscettola 1994b] é um procedimento de colocação de constrangimentos que segue o princípio de flexibilidade temporal. Partindo de um estado inicial correspondente a uma rede de *tokens* (cada um correspondendo a um pedido de capacidade), nenhum dos quais inserido na correspondente *time line* de recurso, tem como objectivo juntar constrangimentos à rede de tokens de modo a que, a inserção de todos eles, de acordo com a rede resultante, não gere conflito de capacidade algum. Para isto, identifica repetidamente conjuntos de *tokens* congestionados (os que têm alta probabilidade de competir pelo uso de um recurso), e junta constrangimentos de precedência para assegurar que não surja nenhum conflito (a identificação destes conjuntos e a determinação de quais os constrangimentos mais favoráveis é feita através de uma metodologia de análise do espaço de procura baseada em simulação estocástica). O algoritmo deste procedimento é o seguinte (estrutura básica do ciclo de resolução de problema idêntica à de outras abordagens de programação heurística):

1. **Análise de capacidade** - Estima a procura de *tokens* e a competição pelos recursos, através de um procedimento estocástico (parametrizado pela estratégia de selecção da variável e pela regra da selecção do valor) em que se gera uma amostra de N atribuições de tempos completas (a todas as variáveis) diferentes. Dada a rede de pontos temporais $<V_t,C_t>$, neste procedimento, repete-se até que todas as variáveis em V_t tenham um valor:
   - Selecciona uma variável de V_t (segundo uma estratégia de selecção de variável prédefinida).
   - Selecciona um valor dentro dos limites temporais actuais (da variável) segundo uma regra estocástica.

- Atribui o valor à variável e propaga as consequências na rede de pontos temporais (o que resulta em novos limites temporais para as variáveis em V_t).
- Retira a variável de V_t.

A partir da amostra obtida calculam-se as seguintes medidas estatísticas de competição e preferência:

Procura de token, $\Delta(\tau,t_i)$ - Para cada *token* τ e cada tempo t_i tal que $EST(\tau) \leq t_i < LFT(\tau)$, é igual a n_{ti} /N, sendo n_{ti} o número de elementos da amostra em que o intervalo de ocorrência do *token* contém t_i. Mede em quanto os constrangimentos e preferências actuais fazem tender o tempo de execução de uma actividade para um determinado tempo.
Competição por recurso, $X(\rho,t_j)$ - Para cada recurso $\rho \in R$ e cada tempo $t_i \in H$ (horizonte de programação), é igual a n_{tj} /N, sendo n_{tj} o número de elementos da amostra para o qual ρ é pedido por mais de um *token* no tempo t_j. Mede em quanto os constrangimentos e preferências actuais gerarão congestionamentos de pedidos de capacidade (e uma inconsistência potencial, portanto) num determinado tempo.

2. **Teste de terminação** - Se a competição por cada recurso é zero, em todo o horizonte de programação, termina com a rede de *tokens* actual como solução.

3. **Detecção de congestionamentos** - Identifica o recurso e tempo pelos quais há maior competição (porção da rede de pontos temporais onde há maior probabilidade de conflitos de capacidade).

   Congestionamento, $<\rho_b,t_b>$ - Dada a competição por recurso $\{X(\rho,t)\}$, com $\rho \in R$ e $t \in H$, é o par $<\rho_b,t_b>$ tal que $X(\rho_b,t_b) = \max\{X(\rho,t)\}$, para qualquer $\rho \in R$ e $t \in H$ tal que $\{X(\rho,t) > 0\}$.

4. **Identificação de conflitos** - Selecciona os *tokens* que mais podem contribuir para o congestionamento (os que potencialmente estarão em conflito).

   Conjunto conflito - É o conjunto de *tokens* que pedem ρ_b, têm limites temporais que contêm t_b, não sendo necessàriamente sequenciais (não é forçoso que dois *tokens* no conjunto conflito sejam adjacentes de acordo com a rede de *tokens*). Havendo vários conjuntos conflito CPS preferirá os *tokens* com perfis de procura perto de t_b.

5. **Partição de conflitos (arbitragem de conflitos)** - Ordena o conjunto de *tokens* segundo a procura de *tokens* inserindo os constrangimentos temporais apropriados, de modo a diminuir a possibilidade de inconsistências na rede de *tokens*. Com o procedimento de arbitragem de conflitos procurou-se um equilíbrio entre introduzir um simples constrangimento de precedência entre pares de *tokens* do conjunto conflito (uma abordagem minimal, com uma filosofia semelhante à da programação micro-oportunística) e impor uma ordem total entre todos os *tokens* do conjunto conflito (uma abordagem de filosofia semelhante à da programação macro-oportunística), para obter um equilíbrio entre a minimização da alteração da topologia da rede de *tokens* e minimização do número de ciclos de resolução do problema (introduzindo demasiados constrangimentos pode gerar inconsistências e aumentar o "backtracking", mas intrdoduzindo poucos em cada ciclo requerirá um maior número de passos de análise de capacidade). Para este equilíbrio a estratégia de arbitragem de conflitos particiona o conjunto conflito em dois subconjuntos, T_{before} e T_{after}, e constrange cada *token* no primeiro de modo a ocorrer antes de qualquer um no segundo. Esta escolha advém da análise dos perfis de procura dos *tokens*, sendo estes atribuídos a subconjuntos de acordo com a aglomeração verificada nos seus perfis de procura.

6. **Propagação de constrangimentos** - Actualiza os limites temporais na rede de pontos temporais (como consequência da introdução dos novos constrangimentos temporais).
7. **Teste de consistência** - No caso de a rede de pontos temporais ficar inconsistente assinala isso e termina.
8. **Continua em 1.**

Se o procedimento terminou saindo por 7. a rede de *tokens* é reposta no estado inicial e o procedimento repetido. O caminho explorado será diferente dada a natureza estocástica da análise de capacidade. No entanto, o CPS terminará por falha se ao fim de um certo número de repetições fixo não se encontrar a solução.

4.5.5. Conclusões e Trabalho Futuro

O autor descreve resultados experimentais em que a heurística CPS se compara favoravelmente a outra heurísticas que não seguem o mesmo princípio de flexibilidade temporal (baseiam-se no compromisso de uma variável com um valor).

Quanto a trabalho a desenvolver no futuro refere a escalabilidade dos modelos de domínio e solucionadores de problemas envolvidos em HSTS, a extensão da linguagem de constrangimentos e dos mecanismos de propagação, o desenvolvimento e avaliação de metodologias alternativas para análise de capacidade, a ampliação das métricas estatísticas do espaço de procura para domínios integrados de planeamento/programação e o teste de eficácia das representações flexíveis de plano/programa durante a execução de planos reactiva.

4.6. GERRY

O projecto de aplicação do sistema GERRY [Zweben 1994] ao problema das Operações em Terra para o Space Shuttle da NASA (programação de reparação e manutenção entre voos) iniciou-se em 1990. A versão inicial do sistema usava um *método construtivo* de programação (extensão incremental de um programa parcial) baseado em *dependency-directed backtracking* na produção de programas com constrangimentos temporais, de datas limite e de recursos, não sendo suficientemente poderosa para manipular de constrangimentos de estado, preempção e capacidades de reprogramação, essenciais para as aplicações na NASA. Assim, o sistema evoluiu para um *método de reparação* iterativo.

4.6.1. Características Gerais

- *Reparação iterativa* - Modificação iterativa de programas completos (possìvelmente imperfeitos).
- Ênfase na *reprogramação* e na *programação preemptiva*.

4.6.2. Aspectos de Modelação

Constrangimentos Temporais

Há constrangimentos temporais *de precedência* que ordenam as tarefas umas em relação às outras (suporta ordenações entre tempos de início, de fim, de início e de fim e de fim e de início, bem como atrasos positivos e negativos na ordem). É usado o algoritmo de Waltz para obter consistência temporal (arco-consistência). Quando há que reprogramar, ou há violação da consistência, deslocam-se outras tarefas até que os constrangimentos sejam satisfeitos.

Há constrangimentos temporais *de datas limite* (milestones) que relacionam as tarefas com tempos métricos fixos, para que elas não sejam deslocadas para lá de certas datas.

Constrangimentos de Recursos

Para cada requerimento de recurso GERRY declara automàticamente um *constrangimento de capacidade*, que indica que não deve haver sobreafectação do recurso durante duração do requerimento.

Classes de recursos representam tipos de recursos, cada uma consistindo num conjunto de *repositórios de recursos* (exemplo: CRANE CREW e HIGH CREW são repositórios de recursos contidos na classe de recursos HEAVY-EQUIPMENT), cada um sendo uma colecção de recursos indistintos. A cada repositório está associada uma *capacidade inicial* e uma *história*, mantida para monitorar a disponibilidade ao longo do tempo.

Para cada tarefa há um conjunto de *requerimentos de recursos*, cada um indicando o tipo (classe de recurso que satisfaz o requerimento) e quantidade de recurso.

Constrangimentos de Estado

Podem modelar-se também *atributos de domínio*, cada um com valores, ou estados possíveis. Cada tarefa tem um conjunto de *efeitos de estado* (que a tarefa impõe nos atributos), cada um com a duração do intervalo de tempo (aberto em caso de persistência indefinida do efeito) especificado pelo efeito. Há *requerimentos de estado* que designam *constrangimentos de estado* sobre as tarefas.

Para cada atributo é mantida uma *história* registando o valor actual do atributo, as tarefas modificadoras (aquelas cujos efeitos alteraram o valor do atributo até ao presente) e utilizadoras (as que têm, e fizeram uso de, um requerimento de estado para o atributo até ao presente) e a útltima tarefa modificadora do atributo, para cada intervalo de tempo (em que há alteração do estado do atributo?) do horizonte de programação, formando uma rede de dependências semelhante à dos sistemas de *truth-maintenance*.

Programação Preemptiva

Cada tarefa é associada a um programa de períodos de trabalho legais e é dividida num conjunto de subtarefas. A programação preemptiva impõe uma sobrecarga computacional porque têm de se calcular os tempos de preempção e realizar a manipulação apropriada dos constrangimentos. Os constrangimentos de recursos e de estado podem ser impostos, alternativamente:

- Durante cada subtarefa individual, sendo ignorados durante os períodos de suspensão, entre as subtarefas (exemplo: mão-de-obra, que estaria disponível nos intervalos entre subtarefas).
- Durante o intervalo de tempo que se estende da primeira à última subtarefas, pelo que deverão ser satisfeitos durante todo intervalo de tempo abrangendo a tarefa (exemplo: maquinaria pesada, difícil de movimentar, estaria não disponível naqueles intervalos).

4.6.3. Procura com Reparação Iterativa

Vantagens

Para GERRY, a reparação iterativa tem as vantagens seguintes:

- **Reprogramação** - O Processamento em Terra para o Space Shuttle é bàsicamente um problema de reprogramação para o que é mais apropriada a reparação iterativa. Para reprogramar usando um método construtivo, o sistema tem de reiniciar o processo de programação removendo antes algumas tarefas do programa, pondo-se o problema de determinar que tarefas remover. Com o método de reparação não é necessário remover tarefas do programa para reprogramar.
- **Oportunidades de reprogramação** - Poderiam perder-se quando tarefas não afectadas de necessidade de reprogramação não são nela consideradas (p.ex., colocar uma tarefa não afectada mais tarde no programa pode causar pouca perturbação e permitir ainda que muitas outras, afectadas, fiquem bem colocadas).
- **Programas superconstrangidos** - No domínio do Space Shuttle, isto acontece, em geral, obrigando, num método construtivo, a testar todas as hipóteses, antes de se concluir que devem relaxar-se constrangimentos. Métodos baseados na reparação tentam iterativamente melhorar soluções e terminam com uma solução tão próxima do óptimo quanto seria possível produzir no tempo dado.
- **Constrangimentos globais e de optimização (critérios)** - Avaliam-se de modo melhor e mais eficiente com métodos de reparação, pois a procura ococrre no espaço de programas completos. Com um programa parcial a avaliação de um critério global só pode ser aproximada.

Desvantagens

As *desvantagens* têm a ver com o facto de os métodos de reparação:

- Poderem ficar presos em *mínimos locais*, perdendo-se em ciclos numa série de soluções não satisfatórias.
- Serem geralmente *métodos não completos*, não garantindo portanto, encontrar a melhor solução.

Simulated Annealing

Técnica estocástica de reparação iterativa usada em GERRY. Um programa inicial completo, de qualidade inferior, é modificado iterativamente (por reparação de constrangimentos violados) até obter uma qualidade satisfatória, ou até ser excedido um certo tempo limite para gerar a solução.

A função de custo, a ser avaliada em cada iteração do algoritmo de *simulated annealing* para medir a qualidade dum programa *s*, é:

$$custo(s) = \sum_{ci \in constrangimentos} penalidade_{ci}(s) * peso_{ci}(s)$$

em que:

- $penalidade_{ci}$ - Uma função de um constrangimento *ci* que produz um número não negativo indicando o grau de violação do constrangimento no programa *s*.
- $peso_{ci}$ - Uma função que dá a importância, ou utilidade, de um constrangimento.

Se:

- O programa gerado numa iteração é melhor (menor valor de custo) que o anterior, ele passa a ser o programa actual (para a próxima iteração); se é também o melhor encontrado até agora é também registado com tal.
- Caso contrário é aceite, ou rejeitado, dependendo de uma probabilidade, que decresce ao longo do tempo/número de iterações (a possibilidade de aceitar soluções piores existe para permitir escapar de mínimos locais e ciclos). Esta probabilidade é dada por uma função:

  $$escape(s,s',T) = e^{-(custo(s)-custo(s'))\ /\ T}$$

  Sendo *T* o parâmetro da "temperatura", gradualmente reduzido durante a procura. Quando uma solução obtida é pior, ela é aceite se um número aleatòriamente gerado, entre 0 e 1, é menor que o valor da função *Escape*.

Resolução de Constrangimentos

A resolução/reparação de constrangimentos violados é feita através de reparações locais invocadas em cada iteração do ciclo de reparação iterativa, reparando-se N violações de cada tipo de constrangimento sucessivamente (o que permite reparar de forma focada num certo tipo de constrangimento).

Há uma forma de reparação para cada tipo de constrangimento violado. Cada reparação é feita sem preocupações de interferência com outros constrangimentos (podendo produzir estados globalmente não satisfatórios que, se aceites, são em geral, melhorados após algumas iterações). Em geral reparar uma violação de constrangimento envolve reprogramação de uma tarefa, sendo deslocadas, se necessário, a antecessora, ou sucessora, da tarefa para manter constrangimentos temporais.

Reparação de Constrangimentos de Recursos

Para cada tarefa envolvida numa sobreafectação de recurso considera-se deslocá-la para um tempo mais cedo, ou mais tarde, mais próximos, em que o recurso esteja disponível. É atribuído um valor heurístico a cada deslocamento candidato, que é convertido numa probabilidade que serve para escolher o deslocamento de tarefa a realizar. O valor heurístico é obtido da combinação linear dos seguintes factores:

- **Ajustamento** - Deslocar a tarefa com requerimento de recurso mais parecido com a quantidade de sobreafectação.
- **Dependentes temporais** - Deslocar a tarefa com o menor número de tarefas dependentes temporais.
- **Distância do deslocamento** - Deslocar a tarefa que obriga a um deslocamento temporal menor.

Reparação de Constrangimentos de Estado

Quando há conflito entre o valor do atributo requerido para uma tarefa e o valor presente considera-se interpor antes da tarefa uma tarefa especial que estabelece o valor de atributo requerido, desde que isso não cause mais violações de constrangimentos de estado, ou então deslocar a tarefa para um tempo mais cedo, ou mais tarde (onde o valor necessário para o atributo existe) ou, em último caso, combina-se o primeiro método com um destes últimos, desde que não se causem mais violações de constrangimentos de estado.

4.6.4. **Conclusões**

O autor descreve testes satisfatórios para este tipo de algoritmos e conclui que a arquitectura de GERRY é a de um sistema de programação poderoso (os testes demonstram que se comporta bem na programação e reprogramação, mesmo para problemas grandes) e independente do domínio (para incorporar um constrangimento novo basta apenas o utilizador definir funções penalidade, peso e de reparação, podendo ainda, opcionalmente, juntar heurísticas de focagem, para ordenar os constrangimentos para reparações).

4.7. CABINS

A originalidade do sistema CABINS [Miyashita 1994] está na aplicação da metodologia de Aprendizagem e Raciocínio Baseados em Casos à revisão de programas.

4.7.1. Características Gerais

- *Reparação iterativa* - Revisão de programas por reparação iterativa no espaço de programas completos. A aplicação da acção de reparação seleccionada resulta na adaptação do procedimento de procura por um desvio nas métricas de competição (pelos recursos) no espaço de procura.
- *Anytime* - O método de reparação (a partir de um programa inicial obtido por qualquer outro método) exibe um comportamento tipo *anytime executable* e o programa resultante é de alta qualidade segundo vários critérios como minimização de disrupções, trabalhos em curso e *tardiness* ponderada.
- *Raciocínio Baseado em Casos* - Usado na aprendizagem e reutilização flexível de preferências de programação e selecção de acções de reparação.
- *Modelo de reparação* - Aprendido incrementalmente e codificado numa *base de casos*, ao contrário de outras abordagens que usam uma única heurística de reparação, ou usam um modelo estatístico prédeterminado para escolha das acções de reparação. A aprendizagem permite uma *escolha dinâmica* das heurísticas de reparação dependendo do contexto de reparação; também, à medida que a base de casos é enriquecida com novas experiências de reparação, o modelo pode adaptar-se a circunstâncias que mudam.
- *Preferências do utilizador* - São reflectidas na base de casos pelas preferências na escolha de uma *táctica de reparação* (dependendo das características do contexto de reparação) e pelas preferências de *avaliação* para o resultado da reparação (resultante da escolha e aplicação de uma táctica específica).
- *Modos de funcionamento* - *Dirigido pelo utilizador* (que escolhe uma táctica de reparação e avalia os resultados da sua aplicação); *assistência interactiva* (o sistema sugere tácticas de reparação e avaliações da sua aplicação, que o utilizador pode, ou não, ignorar); *autónomo* (não há intervenção do utilizador, sendo usada a base de casos adquirida na fase de treino para escolha de reparações e avaliação de resultados delas). Os dois primeiros são usados na aquisição da base de casos (treino).

4.7.2. Aspectos de Modelação e Programação com Constrangimentos

O componente baseado em constrangimentos de CABINS, que serve para produzir o programa inicial, baseia-se em [Sadeh 1990] em que:

- Uma actividade é representada por uma *variável* e uma reserva (tempo de início e conjunto de recursos necessários para uma actividade) é o seu *valor*.
- A *construção do programa* é feita por escolha oportunística de uma actividade a programar e de uma reserva para ela.
- Ao *programar uma actividade* juntam-se os constrangimentos resultantes da nova reserva aos que já existiam e propagam-se. Se na propagação se detecta uma inconsistência

(violação de constrangimento) o sistema faz *backtracking*, caso contrário procede para outra actividade até as programar todas.

- Usam-se *heurísticas* para focar a procura de modo a reduzir o *backtracking* (já que a Programação é NP-difícil) de ordenação de variáveis (determinam que actividade programar a seguir) e de ordenação valores (determinam que reserva atribuir à actividade escolhida).
- Como *heurística de ordenação de variáveis* é usada a heurística ARR (Activity Resource Reliance) que escolhe a actividade mais crítica, isto é, com maior probabilidade de vir a estar envolvida numa violação de constrangimento de capacidade em intervalos de tempo particulares.
- Como *heurística de ordenação de valores* podem ser usada uma de 2 heurísticas: LCV (*Least Constraining Value*) escolhe a reserva que menos impede outras actividades de serem programadas; é caracterizada por uma função de utilidade sem desvio; GV (*Greedy Value*) escolhe a melhor reserva segundo preferências locais, expressas por uma função de utilidade com desvio linear estática. Tempos de início de actividades com altos valores de utilidade são preferidos em primeiro lugar. Segundo [Sadeh 1990]:
  - ARR+LCV - Produz programas suboptimais com backtracking mínimo.
  - ARR+GV (CBS) - Usa funções de utilidade estaticamente prédeterminadas, e é capaz de produzir programas de alta qualidade (apelidado de CBS - Constraint-Based Scheduling).

Nas experiências relatadas [Miyashita 1994] com CABINS, ARR+LCV é usado para produzir um programa inicial (semente).

4.7.3. Procura com Reparação Iterativa

Reparação do Programa

O processo actualmente em reparação é designado por *ordem*, ou *processo focal* e, dentro deste, a reparação é iterada nas suas actividades, designando-se por *actividade focal* a actividade que está em reparação.

No processo de reparação, modifica-se selectivamente o desvio das funções de utilidade dos elementos do *conjunto conflito* da *actividade focal* actual (actividades envolvidas em violações de constrangimentos devido à reparação da actividade focal). O desvio reflecte os efeitos da aprendizagem das preferências e avaliações de resultados de reparações (dependentes do contexto) armazenadas na base de casos e esta modificação dinâmica do desvio afina a procura adaptando-a às características do problema em resolução.

Há dois níveis de reparação:

1 **Estratégias de reparação** - Uma estratégia está associada a uma determinada descrição de alto nível de *classes de defeitos* e com várias tácticas de reparação associadas. Há 2 estratégias de reparação gerais:
   - **Remendo local** - É uma reparação pròpriamente dita, sendo a estratégia por omissão. Em geral envolve menor custo e é menos disruptiva para a operação da fábrica. Exemplo: Se o objectivo de reparação é redução da tardiness reduz-se a folga entre actividades da ordem em atraso e reduz-se o tempo morto dos recursos necessários às actividades da ordem em atraso.

- **Modificação do modelo** - Resultam numa relaxação de constrangimentos temporais, ou de capacidade e originam custos extra (adquirir equipamento/recursos adicionais, mão-de-obra extra ou aumento do número de turnos de trabalho, etc.). Exemplo: Alterar a data limite da ordem em atraso, ou a data de lançamento de ordens (em atraso, ou as que interferem), reduzir a carga na fábrica, aumentar o número de turnos, ou aumentar a capacidade dos recursos.

2 **Tácticas de reparação** - Cada estratégia tem uma colecção de tácticas (acções heurísticas) de reparação associadas. Estas são apropriadas para especializações particulares das classes de defeitos.

Representação de Casos

O CBR de CABINS usa uma *medida de semelhança* para aceder ao caso mais semelhante na base de casos e o do problema actual (ver em [Miyashita 1994] a fórmula da semelhança). Cada caso é indexado pelas características (cada uma com um *valor* e um *peso*, numéricos):

1 **Características globais** - Caracterizam em abstracto a potencial flexibilidade de reparação para todo o programa.
- ***Tardiness* ponderada** -
- **Média de utilização de recursos** - Quando alta indica, em geral, um programa com pouca flexibilidade de reparação.
- **Desvio padrão da utilização de recursos** - Quando alto indica, em geral, muita competição por recursos (e portanto, baixa flexibilidade de reparação).

2 **Características locais** - Associadas ao horizonte temporal de reparação e portanto à actividade focal, prevêm a eficácia da aplicação de uma táctica de reparação particular.
- **Tempo de espera (da actividade focal)** - Intervalo de tempo entre o tempo de fim da actividade precedente na mesma ordem e o tempo de início da actividade focal.
- **Ganho predictivo no deslocamento** - Prevê qual o ganho global, especificamente na redução do tempo de espera da actividade focal, obtido ao deslocar esta para mais cedo, no mesmo recurso, dentro dohorizonte temporal de reparação.
- **Ganho predictivo alternativo no deslocamento** - Idêntico, mas num recurso alternativo.
- **Ganho predictivo na troca** - Prevê a diferença entre a redução do tempo de espera da actividade focal e o aumento do tempo de espera de uma outra actividade, quando se troca no tempo a posição dessas duas actividades, no mesmo recurso da actividade focal e dentro do horizonte temporal de reparação.
- **Ganho predictivo alternativo na troca** - Idêntico, mas num recurso alternativo para a actividade focal.

3 **História de reparação** - Regista a aplicação sucessiva de acções de reparação.
- **Táctica** - A acção heurísticas de reparação.
- **Resultado** - Avaliação atribuída ao conjunto de efeitos de uma acção de reparação (aceitável/inaceitável). Foi indicado pelo utilizador na fase de treino e guiará futuras avaliações.
- **Efeitos: Efeito 1, ..., Efeito n** - Impacto de uma acção de reparação nos objectivos de programação (como *tardiness* ponderada, WIP, etc.), reflectindo tìpicamente compromissos entre vários objectivos.

Aquisição de Casos

O *horizonte temporal de reparação* (da actividade focal) é o intervalo de tempo entre o tempo de fim da actividade precedente da mesma ordem e o tempo de fim da actividade focal e delimita temporalmente a análise da reparação.

No CABINS uma sessão começa com um base de casos vazia. Um conjunto de casos de treino é apresentado ao utilizador e, após a determinação da actividade focal actual, o utilizador escolhe uma táctica de reparação para aplicar, das seguintes:

- **Deslocamento à esquerda** - Mover a actividade focal no mesmo recurso, o máximo para a esquerda na linha temporal, dentro do horizonte temporal de reparação, minimizando a sobreafectação de capacidade resultante.
- **Deslocamento à esquerda em alternativa** - Idêntico, mas num recurso alternativo.
- **Troca** - Trocar a actividade focal com a actividade no mesmo recurso, dentro do horizonte temporal de reparação, que minimiza a violação de constrangimentos de precedência.
- **Troca em alternativa** - Idêntico, mas num recurso alternativo.

Escolhida a táctica pelo utilizador, ela á aplicada ao programa actual e os efeitos da reparação são calculados e apresentados. O utilizador avalia então o resultado. Se a avaliação é positiva (aceitável) o processo é repetido com outra actividade como actividade focal. Se é negativa (inaceitável) CABINS pede ao utilizador uma explicação em termos do peso de cada um dos efeitos, desfaz a reparação, e pede que escolha outra táctica para a mesma actividade focal. Isto é repetido até se ter um resultado aceitável, ou então, quando não restam mais tácticas a tentar com a actividade focal, o processo termina por falha. Na história de reparação do caso são registados: a sequência de aplicações de acções de reparação sucessivas, os efeitos, o resultado, e no caso de falha, a respectiva explicação. Actualmente CABINS já adquiriu, na fase de treino, mais de 4000 casos.

Reparação de Programas Baseada em Casos

Em *modo autónomo* CABINS usa conjuntos de características diversos como índices para acesso aos casos e dá os seguintes passos:

1. CABINS identifica as ordens subóptimas no programa inicial e ordena-as por ordem de suboptimalidade. Na ordem focal as actividades são reparadas para a frente, a começar pela actividade da ordem que tem uma folga acima um limiar heuristicamente determinado (o limiar serve para evitar computação desnecessária e limitar a quantidade de efeitos propagados a outras actividades quando o programa está muito apertado).
2. Cada actividade focal é reparada pelo CBR. No acesso a episódios de reparação anteriores usam-se para indexação o conjunto de características locais e globais. A aplicação da uma táctica descreve-se a seguir.
3. Para avaliar o resultado da reparação depois da aplicação da táctica, usam-se como índices o (tipo de) efeito, o valor e o peso do efeito. CBR é invocado e se o resultado é inaceitável CABINS invoca-o novamente usando como índices a conjunção do resultado actual (inaceitável), a heurística (táctica?) que falhou e as característica locais e globais do caso (para poder, no futuro, aceder a casos que falharam antes da mesma maneira - o uso de CBR no espaço de falhas é um método de recuperação de falhas independente do domínio e permite aceder a soluções para falhas, do passado). Se o resultado é aceitável, CABINS prossegue com a reparação de outra actividade.

Aplicação da Táctica de Reparação Seleccionada

Segue os seguintes passos:

1 Determinação do *tempo de início previsto* para a actividade focal (que pode ser diferente do tempo de início que resulta da reparação), que é uma estimativa temporária usada para calcular os efeitos propagados esperados da reparação em outras actividades (estas são o conjunto de actividades que podem ter de ser reprogramadas devido às violações de constrangimentos resultantes da reparação da actividade focal). É calculado do seguinte modo:
- Para táctica de deslocamento à esquerda (ou idem em alternativa) é igual ao tempo de início que minimiza a sobreafectação de capacidade resultante de mover a actividade focal no mesmo recurso (ou no alternativo) dentro do horizonte temporal de reparação da actividade focal.
- Para táctica de troca (ou idem em alternativa) é igual ao tempo de início que minimiza o número de violações de constrangimentos de precedência no mesmo recurso (ou no alternativo) dentro do horizonte temporal de reparação da actividade focal.

2 Projecção dos efeitos da movimentação da actividade focal para o tempo de início previsto no recurso designado, por propagação de constrangimentos, para identificar violações de constrangimentos.

3 Ajustamento das reservas de todas as actividades envolvidas em violações de constrangimentos (o conjunto conflito da actividade focal) por deslocamento à esquerda, ou à direita, de modo a resolver todos os conflitos.

4 Alterar o desvio das funções de utilidade de tempo de início de todos os elementos do conjunto conflito para favorecer os tempos de início calculados no passo **3.**. Em caso de um recurso alternativo estar envolvido, alterar a função de utilidade do recurso de modo que o recurso alternativo tenha um valor de utilidade mais alto do que aquele em que a actividade focal está actualmente programada. Esta mudança nas funções de utilidade vai reorientar a escolha de tempos de início pelas heurísticas de ordenação de valores (a favor dos que têm maior valor de utilidade, reflectindo as preferências codificadas na base de casos).

5 Desprogramar a actividade focal e todos os elementos do seu conjunto conflito e reprogramá-los usando o programador oportunístico dirigido por constrangimentos com as heurísticas ARR e GV e as funções de utilidade definidas no passo **4.**.

Isto resultará num programa revisto isento de conflitos. Depois, os efeitos da revisão são calculados e CBR será invocado com esses efeitos como índices relevantes para avaliar o resultado da reparação.

4.7.4. Conclusões

Testes foram realizados para comparar CABINS com uma série de heurísticas de despacho largamente usadas em problemas de programação job shop, nomeadamente as regras EDD (Earliest Due Date), WSPT (Weighted Shortest Processing Time), e WSPT com o factores de urgência dos processos (ou regra R & M), e com CBS (Constraint Based Scheduling) com funções de utilidade com desvios estáticos. Para além de superar estes métodos em termos de capturar as preferências do utilizador CABINS produz *programas de melhor qualidade* sem degradação da perfomance segundo vários objectivos de optimização (resultados relativamente à *tardiness* ponderada e WIP são fornecidos em [Miyashita 1994]).

4.8. DAS

O sistema DAS (Distributed Asynchronous Scheduler) [Burke 1991], [Burke 1994] é original na forma de modelar o problema da programação, que aborda segundo um paradigma de *Inteligência Artificial Distribuída*. A distribuição do esforço de resolução do problema é conseguida pelo uso de agentes hierarquizados, reflectindo a organização onde se inserem e a estrutura do problema tratado.

A versão actual de DAS está operando na empresa Alcan Plate Ltd. (APL) em Birmingham, U.K., que produz chapa de alumínio para os mercados aeroespacial e de defesa

4.8.1. Características Gerais

- *Modelação* - Ênfase no aspecto de modelação e representação do ambiente fabril.
- Problema de programação - Entendido como o problema de decidir *o que fazer* (operações), *onde* (recursos) e *quando* (durações, tempos), em princípio com o objectivo de maximizar o lucro da empresa.
- *Satisfação* de constrangimentos - Vê a programação com um problema de satisfação, em vez de um de optimização.
- *Decomposição do problema* - O problema de programação é decomposto e distribuído por um conjunto de *agentes inteligentes*, associados a entidades de uma hierarquia no ambiente fabril.
- Os aspectos *predictivo* e *reactivo* da programação estão integrados.
- *Micro-oportunístico*.
- Combina o algoritmo especializado de propagação de constrangimentos *forward checking* com as técnicas *dependency-directed backtracking* (retrocesso dirigido por dependência) e *shallow learning* (aprendizagem superficial).

4.8.2. Arquitectura

Os Agentes

O problema de programação é distribuído por uma hierarquia (árvore) de 3 níveis de agentes inteligentes autónomos, 1 em cada nó, numa arquitectura fracamente acoplada (baixa razão entre comunicação e processamento).

Os agentes exibem as capacidades reactivas que são necessárias (dada a natureza fortemente acoplada do problema da programação e dado que é necessário aos agentes operarem com informação incompleta e mesmo inconsistente) para operar num ambiente simultâneamente dinâmico e estocástico.

A posição do agente na hierarquia define a sua função e via para comunicação:

- **S-Agent (agente estratégico)** - Introduz trabalho no sistema (o que fazer) e resolve conflitos entre os agentes subordinados. Equivale ao nível máximo de autoridade.
- **T-Agent (agente táctico)** - Delegam tarefas em recursos individuais (onde fazer). Equivale ao responsável por uma área de trabalho dispondo de vários recursos similares, numa fábrica.

- **O-Agent (agente operacional)** - Responsáveis pela execução das tarefas em recursos individuais. Equivale ao chefe de turno que é responsável por um máquina, ou recurso, individual (quando fazer).

O *exterior* pode ser tratado como um outro agente da arquitectura, com a característica de as suas decisões não serem negociáveis.

Funcionamento

Quando um processo vem para o sistema o S-Agent é notificado e *delega* cada operação do plano do processo num T-Agent. Cada um destes, por sua vez, *delega* a sua operação num recurso através do envio de uma mensagem ao O-Agent relevante. Este tenta colocar a nova operação no seu *programa local*:

- Em caso de sucesso - O O-Agent devolve um tempo de início para aquela operação, desencadeando a propagação de constrangimentos através do plano do processo, originando mensagens enviadas a outros agentes (estas têm uma prioridade associada, de modo que o esforço de resolução do problema possa ser focalizado em decisões individuais de O-Agents). Nesta altura não há nenhuma ordenação no plano do processo e as decisões podem ser feitas assincronamente.
- No caso contrário - O O-Agent devolve um *conjunto conflito* (operações que crê não poderem ser programadas consistentemente no seu recurso) ao T-Agent superior. Com a ajuda do T-Assistant este procura equilibrar a carga retirando operações de recursos e delegando operações em recursos. Em caso de não restar opção alguma para equilibrar a carga, conclui que o problema está superconstrangido e devolve ao S-Agent um conjunto conflito (que envolve operações em vários recursos). É o S-Agent que analiza os conjuntos conflito dos T-Agents subordinados e decide o que fazer, que consistirá numa combinação de *backtracking* inter-agentes e relaxação de constrangimentos.

Reactividade, *Backtracking* e Aprendizagem

No decurso destes eventos é obtido novo conhecimento do problema ao nível operacional, táctico e estratégico. Em caso de *backtracking* há informação sobre os caminhos sem saída e em DAS isto é visto como uma *oportunidade para aprender* acerca do problema. Cada agente (a qualquer nível da arquitectura) tem de ter capacidades reactivas, de *dependency-directed backtracking* e de aprendizagem, sendo a capacidade de aprendizagem a que permite reter conhecimento sobre o espaço de procura. A representação usada para recursos e operações permite ao sistema representar todas as oportunidades que restam no programa. O sistema não diferencia alterações induzidas do exterior, ou do interior, bem como não diferencia programação predictiva de reactiva (reacção é uma continuação da procura).

Representação Estrutural

Usam-se técnicas de representação de conhecimento baseadas em *frames* suportadas por KEE, o ambiente de desenvolvimento de software em que DAS foi construído, que permitiu também o uso de *ligação procedimental* e *programação orientada para objectos*.

Representação dos Recursos

Modelam-se recursos da fábrica e centros de maquinação a 2 níveis de abstracção, juntamente com um nível de topo:

1 **Nível de recurso individual (ou operacional)** - O *frame* respectivo contém *slots* que representam:

   – **Preferências de programação** - Como escolher a próxima operação a programar (p.ex., a mais constrangida) e como escolher um tempo de início para uma operação, por exemplo ED (*Earliest Dispatch*), ou JIT (*Just In Time*).

   – **Agente operacional** - O O-Agent associado ao recurso, a sua representação interna do mundo, o *buffer* de mensagens (que origina a actualização daquela representação) e a granularidade temporal.

   – **Detalhes locais do problema** - Operações programadas (afectadas ao recurso e com tempo de início atríbuído) e não programadas (apenas afectadas ao recurso).

   – **Constrangimentos tecnológicos (rígidos)** - Como os de dimensões físicas, de peso, gama de processos possíveis no recurso e também tempos de preparação requeridos por diferentes tipos de trabalhos, ou processos.

2 **Nível de recurso agregado (ou táctico)** - Para além de preferências de programação e do respectivo T-Agent (semelhante ao nível individual), no respectivo *frame* representa-se também uma definição parcial do programa local, através de uma lista de todas as operações que têm de ser processadas por um recurso subordinado mas que ainda não foram delegadas num recurso específico.

3 **Nível de topo (ou estratégico)** - O *frame* respectivo é a raiz da hierarquia e preocupa-se com o problema de programação como um todo, tratando básicamente com planos de processo (enquanto que os restantes níveis tratam com operações). Para além do S-Agent associado e das preferências de programação (que indicam como escolher o próximo plano a delegar no nível táctico) estão também representadas as listas de planos ainda não delegados e de planos parcial ou totalmente delegados no nível táctico.

Operações

Representadas por *frames* com *slots* contendo:

– **Constrangimentos temporais** - Duração, data de lançamento e data limite, relações *before*, *after* e *not.during* com outras operações do mesmo plano de processo.

– **Constrangimentos tecnológicos** - Lista dos recursos capazes de executar a operação e respectivas durações.

– **Tipo** - Pode ser *work* (representa operação normal), *no.shift* (falta de mão-de-obra), *maintenance* (manutenção normal de uma máquina) ou *repair* (falha de máquina). Estas 3 últimas operações são um artifício para ocupar um recurso quando ele não está utilizável, sendo programáveis tal como as do outro tipo (mediante os constrangimentos apropriados).

– **Outros** - Tempo de início e recurso (que definirão o programa), tempos de início válidos (o domínio temporal da operação), a prioridade da operação (a usar pelos T-Agent e O-Agent adequados, não confundir com prioridade de processo), preferência de escolha de tempo de início.

Planos

As relações temporais nos planos baseiam-se na teoria da acção e do tempo [Allen 1984], mas (devido ao oportunismo limitado e à granularidade da representação de recursos limitada -

que não inclui recursos secundários - no domínio de problema actual) bastam apenas as relações *before*, *after* e *not.during* (esta para especificar ordem parcial e, portanto, segundo Fox & Kempf, nestes planos o oportunismo é implicito). Os *slots* respectivos, nas operações do plano, representam os arcos do grafo de operações de um processo. No *frame* do plano há os seguintes *slots*:

1 Operations - Conjunto de nós do grafo de operações.

2 Decision.order - Impõe uma sequência para o lançamento das operações na programação.

4.8.3. A Gestão dos Constrangimentos

O Sistema de Gestão dos Constrangimentos (CMS)

O CMS mantém o domínio temporal de cada operação do problema e informa os agentes quando os seus problemas locais sofreram alteração. A propagação de constrangimentos é feita numa *rede de constrangimentos* em que se representam:

- *Constrangimentos unários*, como tempo de início, recurso, data de lançamento,
- *Constrangimentos binários*, as relações de precedência entre as operações

Estes, combinados, dão origem ao *domínio temporal* (o *slot* dos tempos de início válidos) de uma operação.

Representação de Constrangimentos Temporais

Externamente são um conjunto ordenado de pares de inteiros, cada par representando um intervalo de tempo. Internamente cada uma é composta por uma colecção de constrangimentos componentes (de cuja intersecção se obtém a representação externa), cada um com a forma *(Source (S E))*, em que *Source* identifica a origem do constrangimento componente e *(S E)* o intervalo permitido. Quando uma decisão é desfeita o constrangimento componente a retrair pode ser identificado confrontando essa decisão com *Source*. Também na resolução de conflitos a representação interna de constrangimentos temporais permite a identificação de constrangimentos unários em conflito (pelo S-Agent).

Propagação

A propagação através de constrangimentos binários é originada por uma mudança no domínio temporal de uma operação, resultante da alteração de um constrangimento unário.

Devido à natureza assíncrona da resolução do problema e à dinâmica do ambiente a *retracção de constrangimentos* é usual em DAS. Este processo de retracção é incremental, ao contrário de muitos algoritmos existentes, como o de [Mackworth 1985], que recalculam a rede do princípio quando há uma retracção.

Mensagens

Os agentes mantêm uma visão do mundo e impõem-lhe mudança pelas mensagens que recebem e geram, respectivamente. O CMS é o meio primário de comunicação através de mensagens, que endereça apenas aos agentes a que interessam. Há mensagens para notificar um agente de que o domínio de uma operação do seu problema foi alterado (com uma prioridade indicando a negociabilidade do evento), ou de que uma operação foi adicionada, removida, ou que uma ordem de decisão foi imposta numa operação - atribuição de tempo de início.

4.8.4. A Hierarquia de Agentes

O-Agent, o Agente Operacional

Tem como objectivo manter um programa satisfatório num recurso num ambiente dinâmico/aberto e quando falha (é-lhe permitida uma quantidade limitada de esforço para a resolução do problema) comunica ao seu T-Agent superior uma explicação razoável (uma suposição baseada em conhecimento incerto/incompleto). O que o O-Agent pode fazer:

- Ler mensagens do seu *buffer* e enviar mensagens ao T-Agent superior.
- Ler o *slot* de domínio temporal (tempos de início possíveis) e escrever no *slot* de tempo de início de operações no seu recurso (acção esta que gera, por intermédio do CMS, mensagens).

De acordo com uma *política de comunicação localmente completa* [Durfee 1987] o O-Agent só pode impor um programa ao ambiente quando ele é satisfatório (todas as operações têm tempos de início atribuídos e não há conflitos no programa). Assume-se sempre que está lidando com conhecimento incerto/incompleto (por causa da natureza assíncrona de DAS e da dinâmica do ambiente).

O O-Agent representa o problema através de um *grafo de constrangimentos* em que:

- *Vértices* - Representam operações, sendo o domínio de cada uma um conjunto de intervalos de tempo (o conjunto de tempos de início possíveis da operação)
- *Arcos* - Representam constrangimentos binários.

A programação das operações num recurso é uma instância do *problema de satisfação de constrangimentos dinâmico* e aceita-se uma topologia do grafo de constrangimentos instável. O O-Agent *não* se preocupa com a origem das mudanças devidas a:

- *Efeitos exógenos* - Devidos à incerteza execucional (variações nas durações, ou na chegada antecipada, remoção de mão-de-obra devido ao cancelamento de encomendas ou de processos)
- *Efeitos endógenos* - Devidos à resolução distribuída de problemas (decisões locais e propagação dos seus efeitos, relaxação por alargamento do domínio, ou por remoção de uma operação).

O algoritmo do O-Agent - É um algoritmo híbrido [Prosser 1989], [Prosser 1991], que combina *forward checking* [Haralick 1980] com *shallow learning* [Dechter 1989] e *backtracking* dirigido por dependência [Stallman 1977], ampliado para reagir a mudanças impostas ao agente. Um passo de procura no espaço de estados envolve:

- *Escolha de uma operação* ainda não programada.
- *Atribuição de um tempo de início* do domínio temporal da operação escolhida.
- Aplicação de *forward checking* a partir da operação para todas as operações não programadas e adjacentes à primeira no grafo de constrangimentos. Isto origina a remoção de todos os valores do domínio de cada uma destas últimas operações que são inconsistentes com o valor atribuído à primeira. Os valores removidos ficam explìcitamente representados juntamente com uma justificação (aquela operação e o valor atribuído).
- Se o domínio de uma operação ficar vazio dá-se aprendizagem antes do *backtracking*. Aquela situação acontece por não haver valor algum do domínio da operação consistente com as atribuições das operações em *forward checking* com ela. O O-Agent analisa o

conjunto conflito, constituído por todas estas operações e extrai conhecimento do espaço de estados para guiar o backtracking, limitar o espaço de estados, determinar os efeitos da alteração no seu problema no que respeita ao espaço de procura explorado e obter uma explicação de como se pode relaxar o problema no caso de ele estar superconstrangido.

T-Agent, o Agente Táctico

Básicamente um T-Agent tenta satisfazer constrangimentos tecnológicos de operações associando-as a recursos. Uma operação pode ser vista como uma variável e o seu *slot* de recursos possíveis como um domínio discreto; assim, a delegação de uma operação a um recurso pode ser vista como uma atribuição de valor a uma variável.

Delegação - Uma operação é escolhida, de acordo com o *slot select.strategy* do recurso agregado, por entre as que podem ficar no horizonte de programação requerido. O T-Agent obtém do T-Assistant o conjunto de recursos em que a operação pode ser delegada supostamente sem gerar conflitos a nível operacional (um subconjunto do *slot* de recursos possíveis da operação), escolhe o recurso, faz a delegação e informa o T-Assistant que associa a operação e recurso. Isto tem como resultado uma mensagem enviada pelo CMS ao O-Agent no recurso em causa. O O-Agent envia uma mensagem ao T-Agent informando do sucesso ou falha da programação da operação, passada ao T-Assistant como consequência da decisão feita.

Retracção - Quando o problema ao nível do O-Agent é superconstrangido este envia ao T-Agent superior uma explicação na forma de um conjunto conflito (operações que não podem supostamente ser programadas no seu recurso sem haver inconsistência), que é passada ao T-Assistant. Este devolve uma explicação em forma normal disjuntiva, sendo cada conjunção um conjunto, alternativo, de operações a serem retraídas. Uma delas é escolhida pelo T-Agent para serem removidas dos recursos subordinados. Se o problema ao nível do T-Agent está superconstrangido haverá pelo menos uma operação em cada conjunção que terá de ser removida do problema. O T-Agent envia então ao S-Agent um conjunto de opções em forma normal disjuntiva, em que cada conjunção é um conjunto de operações tal que, se temporalmente relaxadas (domínio temporal aumentado), isso resolverá os conflitos em recursos subordinados.

T-Assistant, o assistente táctico- É um *assumption based truth maintenance system*, em que as operações do T-Agent a que está associado são representadas por variáveis e os recursos por valores, adaptado ao problema de programação táctico dinâmico. Uma atribuição de valor, resultante da delegação de uma operação a um recurso pelo T-Agent, é representada também no T-Assistant por uma assumpção que está IN; um conjunto conflito é tratado como um *nogood* (conjunção de assumpções ilegal); o conselho para resolução de conflitos do T-Assistant é um conjunto de conjunções de assumpções (obtido por hiperresolução negativa nos *nogoods*) tal que, se todas as assumpções de um qualquer das conjuncções forem colocadas OUT, serão resolvidos os conflitos.

Evitando ignorância - Cada recurso operacional é subordinado de um único recurso agregado apenas (e portanto a relação entre T-Agent e O-Agent é uma árvore e não uma rede). Isto evita situações de ignorância total, ou parcial, de um T-Assistant associado a um T-Agent, sobre assumpções correspondentes às operações de conjuntos conflito reportados por um O-Agent comum àquele T-Agent e a um segundo (o T-Assistant poderá não ter alguma, ou mesmo todas, assumpções, por corresponderem a operações delegadas pelo outro T-Agent).

S-Agent, o Agente Estratégico

Aceita trabalho novo que introduz no sistema e delega nos níveis inferiores. É o nível mais alto da hierarquia.

Delegação - Existe sob duas formas:

- **Delegação de um plano inteiro** - Envolve selecção do plano e a delegação das suas operações no nível táctico por associação delas a recursos agregados (o que resulta no envio de mensagens aos T-Agents respectivos, via CMS).
- **Delegação de uma operação** - Quando a tomada de decisão é sequenciada num plano a delegação de operações é desencadeada por decisões ao nível operacional. O S-Agent pode determinar então qual a próxima operação (do plano em questão) a delegar.

Resolução de conflitos - O S-Agent recebe dos T-Agents subordinados explicações de conflitos na forma normal disjuntiva, isto é, conjuntos alternativos de operações que devem ser simultâneamente relaxadas, dos quais escolherá um. Tenta actuar de forma minimal e completa na resolução de conflitos relaxando constrangimentos temporais, tentando primeiro o primeiro dos dois métodos seguintes:

- ***Backtracking* inter-agentes** - É *dependency-directed backtracking* entre T-Agents: se o problema de um T-Agent está superconstrangido devido a uma operação particular, o tempo de início de uma outra operação pode ser removido (para alargar o domínio temporal da primeira) e ela retraída até ao nível estratégico, até que a operação relaxada seja programada. É considerado o impacto de cada operação, de prioridade mais baixa, no domínio temporal da que se pretende relaxar (possível com a representação dual dos constrangimentos temporais) sendo escolhida a operação que, retraída, aumente mais o domínio da operação a relaxar. Para evitar ciclos infinitos na ordem de tomada de decisão neste processo de *backtracking* o S-Agent regista a sequência de decisões implícita resultante.
- **Relaxação de datas-limite** - O S-Agent recorre a isto quando não há operações para retrair, falhando o *backtracking* inter-agentes. Todas as operações do plano/processo são desprogramadas e a seguir reprogramadas da esquerda para a direita no plano do processo. Para compensar o atraso da ordem cuja data limite foi relaxada (ou eliminada) o S-Agent associa, nesta situação, uma estratégia de selecção (de valor) do tipo ED (*Earliest Dispatch*) a cada operação no plano relaxado, com preferência sobre a que está activa no recurso a que a operação foi delegada.

4.8.5. Coordenação do Esforço de Resolução do Problema

O micro-oportunismo em DAS é conseguido através do mecanismo de prioridades.

Prioridade operacional - Atributo que cada operação tem (inicializado a 0), que é uma medida da dificuldade de satisfação dos constrangimentos temporais e tecnológicos associados (pode ver-se como uma história do esforço de resolução do problema dispendido para chegar a uma decisão de programação numa operação).

Sequenciação das tomadas de decisão - Pode ser feita através do mecanismo de prioridade, com maior custo, ou da imposição de uma ordem de tomada de decisão.

Sincronização dos níveis táctico e operacional - Pode haver dois tipos de regimes de sincronização:

- **Disciplinado** - Um T-Agent espera até os O-Agents subordinados tenham resolvido o problema delegado neles, ou chegado a uma situação de conflito, para delegar mais trabalho. Não há assincronia mas o T-Agent obtém a informação completa vinda dos subordinados. Melhor para problemas superconstrangidos.
- **Indisciplinado** - O T-Agent mistura resolução de conflitos com delegação. Pode delegar uma operação num O-Agent subordinado se acredita que ele tenha já resolvido o seu problema ou retrair uma operação se crê que O-Agent precisa de assistência. Maior grau de assincronia, maior posssibilidade de perda de oportunidades na tomada de decisão. Melhor para problemas sub-constrangidos (caso contrário, as decisões tomadas pelo T-Agent basear-se-iam em informação muito incompleta).

Sincronização dos níveis estratégico e táctico - Semelhante à sincronização dos níveis táctico e operacional. O regime indisciplinado resultará possìvelmente num estilo de gestão super-reactivo com perda de informação estratégica, enquanto que no regime disciplinado há perda de assincronia.

Da desordem para a ordem - Na actual implementação de DAS encontrou-se um equilíbrio entre os regimes disciplinado e indisciplinado, originando um trabalho do S-Agent com conhecimento "quase" completo. Para isso só se permite que o S-Agent introduza novo trabalho no sistema se todos os T-Agents já não tiverem trabalho para delegar nos O-Agents subordinados (o que não implica que os T-Agents e os O-Agents tenham atingido um estado final no seu problema, nem que o S-Agent trabalhe com informação completa).

4.8.6. Estado Actual

Na empresa Alcan Plate Ltd., em Birmingham, U.K., a produção está dividida em 3 áreas:

- **Montante** - Produz placas de alumínio.
- **Meio** - Produz chapas por laminagem a quente das placas e rebarbagem e tratamento térmico das mesmas.
- **Juzante** - Acabamento de chapas de alumínio: desempenagem (ST, aprox. 15 minutos por chapa), recozimento (ANN, 18 a 96 horas por lote), tratamento por precipitação (P/T, duração semelhante à de ANN), teste com ultra-sons (US, 2 horas por chapa), serragem final (SAW, 5 minutos por placa, dependendo da máquina usada) e controlo de qualidade e embalagem (I&P, 30 minutos por lote).Tanto ANN como P/T são exectuados com lotes de peças com requerimentos de processamento compatíveis.

DAS está aplicado em juzante, modelando 18 recursos operacionais, com 6 recursos agregados. Há em média 100 processos a decorrer com 4 operações por plano de processo (aproximadamente 400 operações). Há uma ordem parcial envolvida em quase todos os planos, pois P/T e US podem ser trocados. Um plano típico é, por exemplo: {ST *before* {P/T *not.during* US} *before* SAW *before* I&P }

Cada processo representa um lote (um certo número de chapas idênticas com o mesmo plano de processo). Lotes com planos de processo idênticos podem ter durações de processo diferentes devido às diferenças em quantidade de material, ou em tipo de produto. Os tempos de processamento variam muito devido a incerteza operacional (em especial ST).

Interfaces - Há duas: uma permite introduzir processos no sistema (Domain KB, que transforma lotes planeados por uma aplicação possuída pela APL em planos de processo de operações), outra para actualização devida a eventos ocorridos na fábrica (Shop Floor Image

KB, que adquire informação, sobre o progresso do trabalho na produção e a disponibilidade de recursos e mão-de-obra, vinda de uma outra aplicação da APL).

Implementação - Usa Symbolics Common Lisp e KEE para representar recursos, planos e operações. Corre numa máquina Symbolics 3620 com 8MB de RAM e 370MB de disco.

Perfomance típica - Há no total 25 agentes (18 O-Agents, 6 T-Agents e 1 S-Agent) correndo como processos concorrentes na Symbolics. As estratégias de escolha (de valor) são: ST usa ED, P/T, US e SAW usam MID (tenta escolher um tempo de início médio em relação ao domínio temporal), I&P usa JIT.

Actualmente os T-Agents e o S-Agent não têm conhecimento do domínio. Os T-Agents podem delegar trabalho assim que os O-Agents subordinados tenham produzido um programa satisfatório: mistura assim resolução de conflitos com delegação (o que pode reduzir oportunidades para o T-Agent). Também isto acontece com o S-Agent o que pode permitir super-reactividade aos conflitos.

Para um problema de job shop de 90 processos (aprox. 360 operações) DAS produz um programa em menos de 60 minutos. Em média um processo/lote poderá ser introduzido no programa em menos de 1 minuto. Uma avaria de máquina é manuseada em menos de 5 minutos. Espera-se um aumento significativo de eficiência com a introdução de conhecimento do domínio (o que disciplinará entre os níveis a estratégia de resolução do problema) e a distribuição de agentes por vários processadores.

4.8.7. Conclusões

DAS pode ser visto como um sistema tipo *anytime*, pois tenta manter um programa satisfatório num ambiente aberto, em tempo quase real. Também vê a programação com um problema de *satisfação*, em vez de um de optimização. O objectivo maior é satisfazer os constrangimentos temporais e só em caso extremo de impossibilidade opta pela sua relaxação.

A *distribuição do esforço de resolução do problema* é conseguida pelo uso de agentes hierarquizados, reflectindo a organização onde se inserem e a estrutura do problema tratado. Isto permite também uma resolução de problemas distribuída por uma sociedade de agentes, que têm diferentes perspectivas dessa resolução (p.ex., há agentes que tentam minimzar os stocks, usando uma estratégia JIT, enquanto que outros tentam maximizar a utilização de recursos, com uma estratégia ED). Idealmente, se o problema for bem abordado nesta filosofia, pode até ser possível construir sistemas por junção de mais um agente de programação ao sistema. Mas para isto os agentes têm de ser verdadeiramente reactivos (o que o autor crê acontecer em DAS) e capazes de existir numa sociedade democrática (o que parece não acontecer em DAS).

DAS funciona por *delegação de tarefas* (resultantes da decomposição do problema) a agentes especialistas. Uma outra forma que parece ser também promissora é aquela em há uma sociedade de agentes, em que cada agente é capaz de resolver o (mesmo) problema sòzinho, usando técnicas diferentes, com perspectivas diferentes do problema e com diferentes objectivos, havendo entretanto partilha de resultados. Isto pode originar uma aceleração super-linear como a *implosão combinatória* de [Clearwater 1991]. Possìvelmente este processo de distribuição será adequeado quando combinado com técnicas estocásticas de procura (alusão a algoritmos genéticos?).

5. REFERÊNCIAS

Allen 1983 — Allen, J.F., "Mantaining Knowledge about Temporal Intervals", Communications of the ACM 26 (11), 1983, 832-843.

Allen 1984 — Allen, J.F., "Towards a General Theory of Action and Time", Artificial Intelligence, Vol. 23, pp. 123-154, 1984.

Atabakhsh 1991 — Atabakhsh, H., "A Survey of Constraint Based Scheduling Systems Using an Artificial Intelligence Approach", Artificial Intelligence in Engineering, 6 (2) 1991.

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

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

Burke1991 — Burke, P.; Prosser, P., A Distributed Asynchronous System for Predictive and Reactive Scheduling", Artificial Intelligence in Engineering, 6 (3), 106-124, 1991.

Burke 1994 — Burke, P.; Prosser, P., "The Distributed Asynchronous Scheduler" in: Zweben, Monte; Fox, Mark S., "Intelligent Scheduling", 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.

Clearwater 1991 — Clearwater, Scott H.; Huberman, Bernardo A.; Hogg, Tad, "Cooperative Solution of Constraint Satisfaction Problems", Science, Vol. 254, 1181-1183, 1991.

Dean 1987 — Dean, Thomas L.;McDermott, Drew V., "Temporal Data Base Management", Artificial Intelligence, 32 1987, 1-55.

Dechter 1989 — Dechter, Rina, "Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition", Artificial Intelligence, 41 1989/90, 273-312.

Dorn 1994 — Dorn, Jurgen; Slany, Wolfgang, "A Flow Shop with Compatibility Constraints in a Steelmaking Plant", Cap.22, San Francisco, Morgan Kaufman, 1994.

Durfee 1987 — Durfee, E.H.; Lesser, V.R.; Corkill, D.D., "Cooperation through Communication in a Distributed Problem Solving Network", Distributed Artificial Intelligence 1 (ed. M. Huhns), Morgan Kaufman, San Francisco, Calif., 1987.

Fox 1983 — Fox, Mark S., "Constraint-Directed Search: A Case Study of Job-Shop Scheduling", Ph.D. Thesis, Carnegie Mellon University, CMU-RI-TR-85-7, Intelligent Systems Laboratory, The Robotics Institute, 1983.

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

1994.

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.

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

Haralick 1980 — Haralick, Robert M.; Elliot, Gordon L., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence, 14 1980, 263-313.

Kumar 1992 — Kumar, Vipin, "Algorithms for Constraint Satisfaction: A Survey", AI Magazine, 13 (1) 1992: 32-44.

Ladkin 1988 — Ladkin, Peter B.; Maddux, Roger D., "On Binary Constraint Networks", Technical Report, Kestrel Institute, 1988

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 1988a — Collinot, Anne; Le Pape, Claude; Pinoteau, Gérard, "SONIA: A Knowledge-based Scheduling System", Artificial Intelligence in Engineering Vol 3, Nº 2 pp. 86-94, 1988.

Le Pape 1988b — Le Pape, Claude, "Des Systèmes d'Ordonnancement Flexibles et Opportunistes", Ph.D. Thesis, University Paris XI, 1988.

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

Mackworth 1985 — Mackworth, A.K.; Freuder, E.C., "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", Artificial Intelligence, Vol. 25, pp. 65-74, 1985.

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.

Miyashita 1994 — Miyashita, K.; Sycara, K., "Adaptive Case-Based Control of Schedule Revision", in: Zweben, Monte; Fox, Mark S., "Intelligent Scheduling", Cap. 10, San Francisco, Morgan Kaufman, 1994.

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

Muscettola 1994b — Muscettola, Nicola, "An Experimental Analysis of Bottleneck-Centered Opportunistic Scheduling", in: "Current Trends in AI Planning", C. Backstrom and E. Sandewall (Eds.), 1994, IOS Press, Amsterdam.

Prosser 1989 — Prosser, P., "A Reactive Scheduling Agent", Proceedings IJCAI-89, AAAI Press, Menlo Park, Calif., pp. 1004-1009, 1989.

Prosser 1991 — Prosser, P., "Hybrid Algorithms for the Constraint Satisfaction

Problem", Technical Report AISL-46-91, Department of Computer Science, University of Strathclyde.

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

Reis 1996 — Reis, J., "Uma Introdução ao Scheduling", Relatório Interno, Departamento de Ciências e Tecnologias da Informação, I.S.C.T.E., Lisboa, Outubro de 1996.

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

Sadeh 1990 — Sadeh, N.; Fox, M.S., "Variable and Value Ordering Heuristics for Activity-Based Job-Shop", Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Institute of Information Management, Technology, and Policy, University of South Carolina, pp. 134-144, 1990.

Sadeh 1991 — Sadeh, Norman, "Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling", Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1991.

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

Schrag 1992 — Schrag, Robert; Boddy, Mark; Carciofini, Jim, "Managing Disjunction for Practical Temporal Reasoning", Principles of Knowledge Representation and Reasoning, Proceedings of the Third International Conference (KR'92), 36-46, 1992, Morgan Kaufman, California.

Smith 1987 — Smith, Stephen F., "A Constraint-Based Framework for Reactive Management of Factory Schedules", Intelligent Manufacturing, M.D. Oliff (ed.), pp. 113-130, Benjamin-Cummings Publishers, Menlo Park, Calif., 1987.

Smith 1990 — Smith, Stephen F.; Ow, Peng Si; Potvin, Jean-Yves; Muscettola, Nicola; Matthys, Dirk C., "An Integrated Framework for Generating and Revising Factory Schedules", Journal of the Operational Research Society, 41 (6) 539-552, 1990.

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

Stallman 1977 — Stallman, Richard M.; Sussman, Gerald J., "Forward Reasoning and Dependency-Directed Backtracking in a System for a Computer-Aided Circuit Analysis", Artificial Intelligence 9 (1977) 135-196.

Van Hentenrick 1989 — Van Hentenrick, Pascal, "Constraint Satisfaction in Logic Programming", MIT Press, Cambridge, Mass., 1989.

Zweben 1994 — Zweben, Monte; Daun, Brian; Davis, Eugene; Deale, Michael, "Scheduling and Rescheduling with Iterative Repair", in: Zweben,

Monte; Fox, Mark S., “Intelligent Scheduling”, Cap.8, San Francisco, Morgan Kaufman, 1994.

6. OUTRAS FONTES

Reis 1995 Reis, Joaquim, “Gestão de Recursos e Tempo Usando Técnicas de Inteligência Artificial”, projecto de tese de doutoramento (PROJECTO.DOC), I.S.C.T.E., 25-Jan-1995.