4 Planejamento e Escalonamento Integrados

Como a meta deste trabalho é obter uma solução que ofereça ao mesmo tempo flexibilidade e integração, a essência da abordagem proposta é trabalhar com uma arquitetura de dois níveis, integrando as restrições básicas do problema e os componentes do plano, que de forma incremental irão construindo uma solução válida, intercalando o planejamento com o escalonamento.

Este capítulo apresenta um algoritmo que oferece uma solução para o problema de flexibilização temporal em planos através da integração da fase do planejamento com a fase da execução. A seção 3.1 introduz o problema em mais detalhe. A seção 3.2 apresenta algumas abordagens encontradas na literatura para a solução do problema e indica a que foi escolhida na dissertação. Tomando como base os trabalhos de [Barber, 2000], [Mieri, 1996] e [Detcher, 1991], [Afonso, 2004] e [Garrido, 2000], a seção 3.3 inicialmente descreve uma proposta de arquitetura. Em seguida, discute a escolha de um algoritmo como ponto de partida para a implementação de uma solução do problema de flexibilização temporal em *workflows*. Por fim, apresenta o algoritmo com as alterações necessárias para a introdução do módulo de escalonamento. A seção 3.4 contém as conclusões deste capítulo.

4.1 Introdução

Na modelagem e especificação de um *workflow* temos que considerar as restrições de tempo, sejam elas fruto de indisponibilidades, falhas ou quaisquer outras ocorrências que prejudiquem a execução das ações. A abordagem clássica para a solução deste problema trabalha com a decomposição das ações em duas etapas distintas, de forma seqüencial, tratando cada uma delas como um problema independente: na primeira etapa, são selecionados e organizados os conjuntos de ações para alcançar as metas desejadas, o que resulta em um plano; na segunda

etapa, é feito o escalonamento das ações no tempo de execução a alocação dos recursos disponíveis.

Esta separação entre geração do plano e escalonamento (temporal) das ações gera alguns inconvenientes. Por exemplo, se uma inconsistência é identificada durante a etapa de escalonamento, então é necessário retornar à etapa de planejamento para ajustar o plano, considerando as novas restrições, o que muitas vezes leva até a interrupção do workflow. A solução proposta neste capítulo procura lidar com este tipo de situação através da integração entre planejamento e escalonamento.

Assim, a partir de uma análise preliminar, podemos relacionar as principais características que o algoritmo integrado deverá ter:

1- Tratar ações com duração distinta e gerar planos em que as ações possam ser executadas de acordo com as restrições temporais identificadas;

2- Separar o problema de determinar as ações que são necessárias para alcançar os objetivos relevantes do problema de escalonar as ações de forma a garantir a disponibilidade dos recursos (*abstração de recursos*). O plano abstrato assim gerado deve ser independente dos recursos existentes, o que facilita a escalabilidade do sistema, principalmente nos problemas com elevado número de recursos. Uma vez obtido o plano abstrato, um escalonador deve ser encarregado de alocar os recursos necessários para executar as ações segundo a disponibilidade.

3- Reduzir o espaço de busca fazendo com que os objetivos a serem satisfeitos interfiram o mínimo possível na abstração da gestão destes recursos.

4.2 Abordagens para Integração de Planejamento e Escalonamento

As técnicas para solução de problemas de planejamento e de escalonamento são fundamentalmente diferentes. Se, por um lado, o planejamento

lida com as dependências causais existentes, derivando destas o conjunto de ações a serem executadas, por outro lado, escalonamento envolve a propagação de restrições.

Ambos tem em comum o fato de utilizarem algoritmos de busca com técnicas específicas. Podemos ainda identificar duas vertentes distintas, do ponto de vista de integração entre o planejador e o escalonador: acoplamento fraco e acoplamento forte. A discussão sobre qual tipo é mais adequado basicamente está relacionada à forma como informações são trocadas entre os módulos, aumentando o grau de flexibilização, o que irá ser fundamental na escolha do algoritmo de referência.

4.2.1 Arquitetura com acoplamento fraco

Nesta arquitetura os sub-problemas de planejamento e escalonamento são resolvidos separadamente. De acordo com as atividades a serem executadas e as metas a serem atingidas, o planejador gera um plano atemporal, com a descrição das relações de precedência entre as atividades. Toda informação relacionada ao tempo e à utilização de recursos é removida do domínio, ficando sob a responsabilidade do escalonador. O plano resultante é normalmente um *plano de ordem total* (POT), ou seja, uma seqüência de atividades (operadores instanciados).

Uma desvantagem desta abordagem está ligada ao fato de que nem toda precedência entre as atividades necessita ser mantida, pois algumas podem ser executadas em paralelo. Portanto, um plano de ordem total deve ser transformado num *plano de ordem parcial* (POP) através da execução de *algoritmos de relaxamento da ordenação*, como os descritos em [Backstorm, 1998] e [Moreno, 2004]. Numa outra abordagem encontrada em [Cesta, 1999], são utilizados algoritmos que geram POTs diretamente, diminuindo a comunicação e a flexibilidade do *workflow*.

Deve ser observado também que, no caso de ser identificada uma inconsistência durante a execução do plano, não é possível a correção da inconsistência, pois não existe uma integração que permita este tipo de operação de forma simples (figura 10).

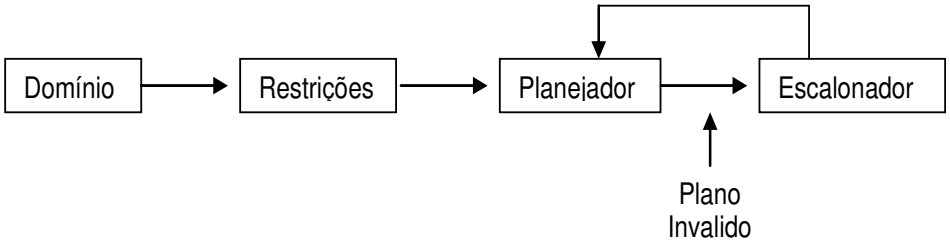

Figura 10 - Arquitetura com acoplamento fraco

4.2.2 Arquitetura com acoplamento forte

Nesta arquitetura, (figura 11), os problemas de planejamento e escalonamento são reduzidos a uma representação uniforme, sem a decomposição em dois problemas seqüenciais distintos, conforme descrito na arquitetura anterior. A vantagem que esta arquitetura apresenta está na idéia de que, ao reduzir o planejamento e o escalonamento a um único formalismo, facilita-se o desenvolvimento de um único algoritmo para execução do workflow, tirando vantagem das informações contidas e compartilhadas durante a execução.

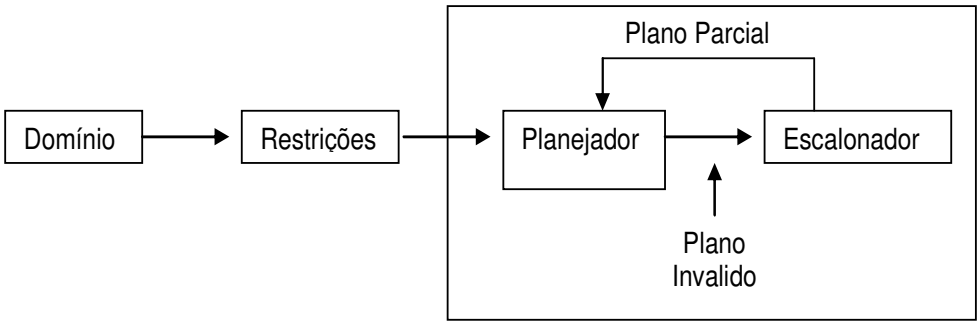

Figura 11 - Arquitetura com acoplamento

4.3 Integração de Planejamento e Escalonamento

4.3.1 Arquitetura de Integração

Do ponto de vista do planejamento, o processo de busca considera uma visão mais abstrata, com foco na definição e representação de metas (G) e

operadores (O), estabelecendo os relacionamentos entre eles, já que as restrições, condições temporais e os recursos serão tarefas delegadas ao escalonador.

Na modelagem proposta, o processo de planejamento deverá procurar otimizar alguma métrica. Assim, a cada escolha, será realizada uma consulta ao escalonador, que verifica a consistência entre as restrições e condições de tempo. Esta integração possibilitará, caso seja detectada uma inconsistência, o reparo do plano.

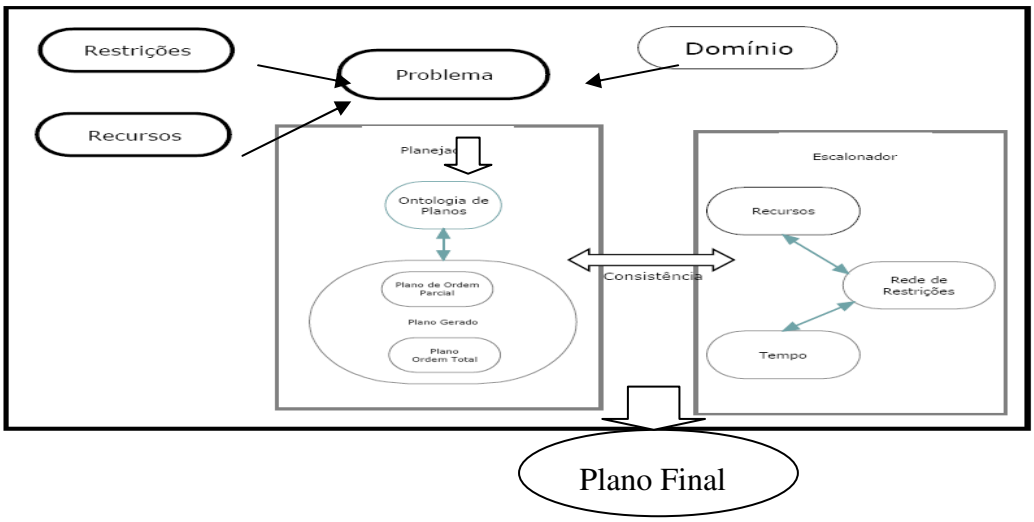

Figura 12 - Arquitetura proposta para a integração

A Figura 9 demonstra esquematicamente o funcionamento do algoritmo de integração. Como informações iniciais, é necessário que sejam definidos o domínio do problema, as restrições, os recursos inicialmente disponíveis e a meta a ser alcançada. A partir destas informações, é feita uma consulta à ontologia de planos com o objetivo de identificar possíveis soluções. Caso algum plano candidato seja encontrado, este servirá de ponto de partida para a geração do plano de ordem parcial. No próximo passo, a consistência do plano será testada pelo escalonador, através do uso da ontologia de recursos e das restrições temporais métrico-disjuntivas do sistema. Caso seja identificada alguma inconsistência, o algoritmo procura uma solução, seja replanejando, seja construindo um novo plano.

A questão da integração do planejamento com o escalonamento já foi estudada em diversos trabalhos. Grande parte das soluções são baseadas em algum tipo de planejamento temporal, ou seja, em algum algoritmo para satisfação de restrições temporais ([Smith, 1996], [Laliberty, 1996], [Smith, 1993], [Tsamardinos, 1988], etc.). Estas soluções oferecem resultados mais restritos, pois como consideram somente as restrições temporais, sem considerar por exemplo as restrições de recursos ou de equipamentos, levam a diminuição do espaço de busca. Podem ser citados como exemplo os seguintes sistemas que utilizam esta estratégia: O-Plan [Currie91], SIPE [Wilkins95], e ParcPlan [Kholy96]. Na abordagem proposta, utilizaremos como referência o QPRODIGY [Borrajo, 2001], ao qual inseriremos algumas modificações, no algoritmo base, para que seja possível tratar questões relacionadas ao escalonamento de tarefas.

4.3.2 Algoritmo de planejamento

Na modelagem do problema de planejamento, podemos destacar o trabalho desenvolvido na Universidade de Camegie Mellon, que utiliza métodos de escalonamento baseados em restrições, além de heurísticas, para otimizar a busca de soluções para o problema. Os métodos são aplicados a domínios onde existe a necessidade de uma solução integrada entre o planejamento e o escalonamento. Como resultado do trabalho, foi desenvida a arquitetura HTST [Muscettola, 1994] , que contempla a integração como um processo de adição de restrições. [Bartak, 2000] apresenta outro trabalho que propõe um estrutura geral de escalonamento com certa capacidade de planejamento, considerando a iteração entre tempo e recursos.

Neste trabalho, utilizaremos como referência inicial o QPRODIGY[Borrajo, 2001], que apresenta as características básicas listadas anteriormente necessárias para o tratamento de restrições. O algoritmo é uma versão não linear do PRODIGY [Veloso, 1995], utilizando os mesmos pontos de decisão na árvore de busca. A linguagem de representação utilizada é uma versão aumentada da ***St****anford* ***R****esearch* ***I****nstitute* ***P****roblem* ***S****olver* [STRIPS], que incorpora características da ***P****lanning* ***D****omain* ***D****efinition* ***L****anguage* [PDDL 2.2], tais como uma linguagem de controle do conhecimento ou funções para operadores de restrições. Estas características permitirão uma melhor

especificação do problema, bem como lidar com as métricas relacionadas com cada operador do domínio do problema. Caberá ao usuário definir a lista de métricas e computar o valor para cada operador. Assim, não existe uma métrica pré-definida, dado que cada usuário pode definir sua própria métrica de acordo com as características do problema, aumentando a flexibilização.

O processo de busca é realizado bidirecionalmente e em profundidade, explorando o espaço de soluções, do estado atual S até a meta G e da meta G até o estado S. A eficiência da busca depende das escolhas realizadas nos chamados pontos de decisão, onde são incluídas regras de controle, usualmente dependentes do domínio. As regras de controle são normalmente estruturas do tipo IF-THEN-ELSE, que informam ao sistema quais escolhas devem ser realizadas ou evitadas, dependendo do estado corrente **C** em que se encontra o processo, das pré-condições não alcançadas, e de outras informações baseadas em escolhas prévias ou em sub-metas a serem atingidas.

O algoritmo segue um ciclo de decisão com fases distintas. Em uma primeira fase, o sistema deve decidir onde aplicar os operadores do planejamento que são pertinentes à situação corrente ***S***. Isto é feito realizando uma busca para frente, certificando-se de que cada operador a ser aplicado está de acordo com o custo esperado da operação, ou realizando uma busca para trás selecionando uma meta não alcançada $g \in G$. O próximo passo deve ser a escolha de um operador ***O*** capaz de satisfazer a meta ***g*** escolhida.

Fica claro que os pontos de decisão são, na realidade, pontos de referência para a realização de *backtracking*. Eles permitem selecionar, rejeitar ou escolher alternativas através da definição de regras de controle. O objetivo final da seleção e rejeição é diminuir o espaço de busca; por outro lado as regras de preferência determinam quais partes devem ser exploradas.

No primeiro ponto de decisão, o algoritmo aplica as regras selecionadas cuja condição seja satisfeita pela situação corrente. Cria como resultado um conjunto de candidatos no espaço de busca. No próximo ponto de decisão, o algoritmo aplica as regras de rejeição sobre o estado corrente, realizando uma poda nos candidatos. Finalmente, o algoritmo aplica as regras de controle, que

seguem uma hierarquia preferencial, para determinar a ordem de exploração dos ramos candidatos remanescentes, resultantes das etapas anteriores.

O algoritmo **QPRODIGY** [Borrajo, 2001] é apresentado na próxima pagina:

QPRODIGY (S,G,M,B,C,P,O, O_b,T,g,b)

Definição

Gerar plano

Entrada

S = Estado inicial
G = Metas
M = Métrica de custo
O = Nome do operador
O_b = Operador instanciado
T = Tempo
B = Cost Bound
C = Control knowledge
g = Meta final
b = Binding

Saída

P = Plano

IF T= 0

THEN retorna P

IF $G \subseteq S$

THEN

IF várias soluções = verdadeiro

THEN $B \leftarrow \sum_{O_j \in P}$ custo (O_j, M)

Backtrack (continua busca com novo limite para o custo)

ELSE retorna P

/* escolhe "Busca para frente" ou "Busca para traz" de acordo com C */

IF "Busca para frente"

THEN selecione O_b utilizando C tal que

as precondições de O_b sejam verdadeiras em S

IF custo (O_b, M) + $\sum_{O_j \in P}$ custo (O_b, M) < B

THEN S = aplicar (O_b, S);

compute novamente G;

$P = P + O_b$

ELSE *backtrack*

ELSE /* Busca para traz */

Selecione meta g ∈ (G – S) utilizando C;

Selecione operador O que satisfaça g utilizando C;

Selecione um *binding* b para O utilizando C;

Recompute novamente G;

QPRODIGY (S,G,M,B,C,P,O, O_b,T,g,b)

As alteraçoes inseridas no QPRODIGY [Borrajo, 2001] foram a inclusão de regras de controle, onde as restrições temporais serão tratadas através do módulo de escalonamento. O algoritmo resultante, será chamado de Algoritmo de Integraçao Planejamento/Escalonamento (AIPE) sendo apresentado a seguir:

Passo 1 – Seleção da meta.

Na primeira ação é selecionada uma meta do conjunto de metas e submetas não resolvidas. Neste passo é verificado o limite de tempo (métrica de custo) para a execução da busca da solução, caso o custo estimado para a solução tenha sido ultrapassado, o algoritmo retorna o plano P.

Passo 2 – Seleciona um operador em que as precondições coincidam com a meta selecionada.

A escolha de um operador que tenha metas semelhantes ou coincidentes com a meta selecionada no passo 1, irá permitir a diminuição do custo e espaço de busca. São então instanciadas todas as possíveis variáveis neste operador, caso este possa ser aplicado, ele poderá ser parte da solução, caso contrario não e o algoritmo escolhe outra meta.

Passo 3 – Processo de busca

O algoritmo inicia um proceso de busca para frente ou para traz até que seja encontrada a solução para a meta escolhida, ou retorna uma solução caso o tempo limite fornecido para a busca expire, ou ainda caso o número máximo de nós seja excedido ou a árvore de busca seja toda pesquisada.

Passo 4 – Executa módulo PSR

O algoritmo chama o módulo PSR para verificar a consistência da solução encontrada e continuar o processo de busca da solução. Realiza a verificação da consistenca da solução.

AIPE (S,G,M,B,C,P,O, O_b **,T,g,b)**

Definição

Gerar plano

Entrada

S = Estado do problema
G = Conjunto de metas a serem alcançadas (metas pendentes)
D = Descrição do domínio do sistema (operadores)
P = Plano inicialmente vazio
M = Métrica de custo
O = Operador instanciado (do conjunto de operadores em *D*)
B = substituição de variável em um operador
O_B = Operador *O* instanciado com variável substituída
○ = Conjunto de operadores instanciados não pertencentes a *P*, vazio
A = Conjunto de operadores aplicáveis (subconjunto de O_B)
G_0 = Conjunto de precondições de todos os operadores em ○
ST = Árvore de busca
LE = Conjunto de links em *P*
L = Links para cada O_B
LR = Links para cada O_B adicionado por conflito de recursos
T = Tempo
B = Custo máximo
C = Conjunto de regras de controle

Saída

P = Plano

IF tempo total excede tempo limite **THEN** retorna *P* // Passo 1

IF $G \subseteq S$

THEN IF múltiplas soluções = verdadeiro // Passo 2

THEN $B \leftarrow \sum_{O_j \in P}$ custo (O_j, M)

Backtrack(continua busca com nova métrica de custo *M*) // Passo 3

ELSE retorna *P*

ELSE /* Escolhe "Busca para frente" ou "Busca para traz" de acordo com *C* */

IF realiza busca para frente

THEN selecione O_B utilizando *C* tal que

as precondições de O_B são verdadeiras em S

IF custo (O_B, M) + $\sum_{Oj \in P}$ custo (O_B , M) < B // Passo 4

THEN atualize P aplicando O_B, desordenar o plano P e atualize móduloPSR com novos precedentes

IF P = consistência temporal (dentro do limite de tempo para busca da solução)

THEN computar novos precedentes móduloPSR

IF P = consistente com os recursos disponíveis

THEN S = aplicar (O_B, S) nova iteração em G,$P = P + O_B$

ELSE falha

ELSE falha

ELSE realiza o *backtrack*

ELSE (*) Selecionar meta $g \in (G - S)$ utilizando C

(*) Selecionar operador O que satisfaça g utilizando C

(*) Selecionar um *binding* b para O utilizando C

Atualizar metas e continuar G

AIPE (S,G,M,B,C,P,O, O_B,T,g,b)

4.3.3 Algoritmo de Escalonamento

O módulo de escalonamento proposto terá caráter iterativo, tratando as restrições temporais de forma sucessiva, organizadas em uma rede de restrições. A solução procura limitar e restringir a propagação das restrições, o que possibilitará manter um conjunto mínimo de soluções durante todo o processo de busca da solução. A idéia central é conjugar as vantagens da propagação de restrições, ou fechamento [closure] com as vantagens do modelo de Satisfação de Restrições (*SR)*, discutidas no trabalho de Detcher [Detcher, 1991], onde procuramos construir a solução mediante a instanciações sucessivas de variáveis.

A abordagem proposta assume um procedimento de busca com *backtracking,* onde a obtenção da solução é feita mediante uma seleção sucessiva de pares de variáveis. E esta abordagem segue o conceito de ramificação e

posterior poda onde vão sendo ordenadas as tarefas que compartilham o mesmo recurso.

Como resultado obtemos um modelo de resolução que incorpora as restrições temporais e permite a flexibilização do *workflow*, tratando os erros e as exceções, quando aplicado a problemas de planejamento e escalonamento. Outra característica importante que este módulo incorpora é o tratamento e a adição de novas restrições que são obtidas com novas informações obtidas da rede durante a execução, o que poderá permitir a redução do espaço de busca das soluções.

A seguir é feita uma descrição de cada etapa do módulo de escalonamento, e posteriormente o modulo em si é apresentado.

Passo 1 - Definição do Tipo da restrição.

O primeiro passo do algoritmo consiste em determinar o conjunto de restrições temporais (t_i c_{ij} t_j) associadas ao problema de escalonamento, expressadas segundo a linguagem proposta no item 3.3.7.

As especificações são traduzidas, fazendo a conversão dos dados de entrada em restrições temporais métrico-disjuntivas, sobre as quais podem ser aplicadas o algoritmo de restrição. Este processo gera um conjunto de restrições temporais C= {(t_i c_{ij} t_j)} que podem ser divididas em dois grupos:

1 - subconjunto de restrições não disjuntivas, formadas por restrições de precedência, duração, tempos de começo e finalização e

2 - subconjunto de restrições disjuntivas, caracterizadas pelas restrições formadas pelos intervalos temporais.

Passo 2 - Restrições não disjuntivas.

Dado o conjunto de restrições temporais C = {(t_i c_{ij} t_j)}, é dada preferência ao processamento do subconjunto de restrições ***não disjuntivas***.

Quando selecionamos e propagamos as restrições, se por um lado, o custo de propagação é menor, por outro lado, permite detectar inconsistências antes que ocorram. Quando é realizada uma atualização na rede temporal com uma restrição não disjuntiva, é possível detectar as inconsistências e determinar se o problema tem ou não solução. A propagação é realizada utilizando um algoritmo básico de fechamento apresentado no item a seguir. Como resultado do processamento das restrições não disjuntivas, obtemos uma rede com restrições binárias formadas por um único intervalo temporal.

Passo 3 - Regras de exclusão.

Previamente ao processamento do subconjunto de restrições disjuntivas, são aplicadas as regras de exclusão de [Caseau94] como pré-processo, antes de utilizar as heurísticas de ordenação de variáveis. Como resultado da aplicação destas obtemos um conjunto de restrições não disjuntivas, a partir das restrições disjuntivas pendentes de serem seqüenciadas. Somente usaremos as regras quando na rede não tenhamos restrições disjuntivas, e têm por objetivo evitar uma possível realização de *backtracking* sobre o conjunto C de restrições disjuntivas pendentes.

Esta etapa tem por finalidade determinar se uma atividade o_i que é realizada sobre um certo recurso M_i deve ser realizada antes ou depois de um conjunto de atividades já seqüenciadas sobre o mesmo recurso. Para tanto, para cada uma das restrições disjuntivas pendentes do conjunto C, são analisadas se é possível os seqüenciamentos que representam a disjunção. Como resultado do processo, possivelmente algumas das restrições disjuntivas pendentes se convertem em não disjuntivas. Considerando que cada restrição disjuntiva expressa um possível seqüenciamento entre duas operações que competem por um mesmo recurso, neste caso as regras de exclusão permitem que seja feita a escolha, entre considerar o_i antes de o_j, ou o_j antes de o_i. Desta forma reduzimos o risco de ser necessário a realização de um *backtracking* em passos posteriores como conseqüência de uma escolha errada do escalonamento. Se for detectada alguma inconsistência durante este passo, podemos dizer que, dada uma restrição

disjuntiva, podemos deduzir então que não é possível a execução de nenhuma das seqüências, e logo o problema é inconsistente, não sendo possível encontrar nenhuma solução.

Passo 4. - Cota superior do espaço solução e regras disjuntivas.

Nesta etapa são realizadas ações de forma repetitiva mesmo que estas apresentem restrições disjuntivas que estejam pendentes de seqüenciamento.

Nesta etapa em primeiro lugar aplicamos uma heurística global para limitar o final do escalonamento, reduzindo a árvore de busca. Assim conseguimos obter durante todo o processo os possíveis espaços solução mínimo e o máximo, através do conjunto de restrições já identificadas (para os quais devem ser consultadas as restrições existentes entre os pontos da rede ***T0*** e ***TF***).

Isto ocorre porque, como conseqüência do processo de propagação das restrições, são deduzidas e acrescentadas novas restrições, de forma que podemos conhecer as restrições entre quaisquer pontos da rede. Este procedimento é utilizado para a obtenção de novos limites do espaço solução, levando em conta as restrições disjuntivas que ficaram pendentes de serem tratadas. Para isso será utilizada uma medida de textura baseada em informações globais e que como resultado levará a consideração de uma nova restrição $C_{T0.TF,}$ cuja propagação utilizará este espaço solução como cota superior. Este procedimento limita os domínios dos valores das variáveis do problema, e reduz o custo computacional da propagação.

A conseqüência de restringir o final do escalonamento, é que, em geral, cada vez que uma nova restrição é inserida, outras também serão, e que algumas das decisões disjuntivas pendentes serão convertidas em não disjuntivas. Para detectar estas novas "não disjunções" que aparecem, são então aplicadas ***regras disjuntivas.*** Se depois de propagar as novas restrições não disjuntivas, forem detectadas inconsistências, então, e dependendo da existência de alternativas de seqüenciamento, será realizado um processo de *backtracking*. Se for detectado uma inconsistência, e não se dispõe de seqüenciamentos, não realizados e

previamente armazenados, então o processo termina, já que não é possível encontrar uma solução.

Passo 5 - Heurística de seleção de variáveis.

Na 5ª etapa serão tratados os subconjuntos de restrições disjuntivas. Cada uma delas implica em uma decisão de seqüenciamento das operações o_i *e* o_j, que competem pelo mesmo recurso.

Podemos considerar que cada restrição disjuntiva pendente é uma variável, e que cada seqüenciamento possível $o_i \rightarrow o_j$ ou $o_j \rightarrow o_i$, ou como os valores possíveis para dada variável. Por tanto, para eleger a seguinte restrição disjuntiva a ser processada, que representa o próximo par de operações a ser seqüenciados, utilizará uma heurística de seleção de variáveis dadas as operações não ordenadas o_i *e* o_j. O cálculo temporal para as possíveis ordenações, nos proporciona a base para identificar a seqüência mais *crítica*, ou seja a mais restrita. Também estaremos tratando da ordenação de valores, ou seja decidir, se dado um par de operações não ordenadas, um seqüenciamento, dos possíveis pares. Para isso será dada preferência para a que oferece o maior grau de liberdade.

A utilização de heurísticas combinadas permite uma melhor seleção das variáveis, a idéia da utilização consiste em aplicar duas ou mais heurísticas e estabelecer um critério de seleção com base em todas elas. Por exemplo, suponhamos a seguinte situação: ao aplicarmos uma heurística de seleção de variáveis H_1. Como resultado de H_1 obtemos um conjunto de restrições disjuntivas. Neste caso não existe uma única escolha possível, pois todas conduzem a uma solução do problema.

Utilizamos também uma heurística de ordenação de variáveis H_2, e como resultado obteremos também várias alternativas possíveis. Para tomar uma decisão iremos realizar a interseção dos conjuntos resultantes de aplicar H_1 e H_2 existindo duas possibilidades:

• O conjunto interseção não é unitário, porque provavelmente o número de alternativas será menor. Em qualquer caso, estamos "reforçando" a decisão de que não nos baseamos em uma única heurística. Quer dizer, melhoramos a qualidade da informação que dará como resultado a eleição da seguinte restrição disjuntiva que será processada, ou qual é o seguinte par de operações (o_i,o_j) para seu seqüenciamento.

• Se a interseção não contém elementos, significará que cada heurística detecta operações distintas. Então podemos dar prioridade ao resultado de um dos critérios de eleição frente ao outro, ou simplesmente eleger a seguinte restrição disjuntiva do conjunto *C*. Em qualquer caso, obteremos como resultado a seguinte restrição disjuntiva que deverá ser processada.

Passo 6 - Heurística de seleção de valores.

Uma vez obtido o próximo par de operações (o_i,o_j) a serem seqüenciados, faremos a escolha da restrição disjuntiva associada, propagando esta a todos os arcos da rede mediante o algoritmo de fechamento

A restrição c_{ij} é convertida numa restrição disjuntiva etiquetada lc_{ij} imediatamente antes da propagação, as etiquetas associadas aos intervalos, indicando as ordenações possíveis, isto é, sendo os intervalos temporais correspondentes ao dos seqüenciamentos possíveis $o_i \rightarrow o_j$ ou $o_j \rightarrow o_i$. O processo de propagação pode detectar uma inconsistência, que iniciará um processo de *backtracking*, considerando as restrições armazenadas e encontradas anteriormente. Se não forem detectadas inconsistências, então, como resultado da propagação, entre cada par de pontos da rede existirá uma restrição formada por intervalos temporais, cada um dos quais irá conter entre suas etiquetas $R_{i\rightarrow j}$ ou $R_{j\rightarrow i}$. Se são detectadas devemos decidir qual ordenação será realizada, $o_i \rightarrow o_j$ ou $o_j \rightarrow o_i$, para o qual usaremos uma heurística de ordenação de valores especificadas no parâmetro *valores*.

Novamente podemos utilizar uma única heurística ou então combinar o resultado de varias delas, ainda que a diferença fundamental com a escolha de variáveis é a capacidade de manter as indecisões:

• Aplicamos as heurísticas escolhidas, e comparamos os resultados derivados de cada uma delas. Dependendo do critério adotado se realiza uma escolha. A solução eleita será utilizada, implicando que as restrições descartadas serão armazenadas pois em caso de ser necessário a realização de um novo processo de *backtracking* que desfaça a escolha, estas estarão armazenadas, o que diminui o espaço de busca e o custo computacional .

• Pode ocorrer também que, como resultado da aplicação das heurísticas anteriores, não seja possível tomar nenhuma decisão, como por exemplo se cada heurística resulta em uma escolha distinta. Neste caso, é possível não tomar nenhuma decisão e manter essa disjunção na rede, já que o algoritmo permite manejar disjunções com cardinalidade maior do que 2. Então se mantemos a disjunção, será realizado o armazenamento em um conjunto que denominaremos ***CInd*** da restrição (t_i c_{ij} t_j), assim como as etiquetas $\{R_{i \to j}, R_{j \to i}\}$ associadas a estas restrições.

A manutenção de indecisões representa uma novidade com relação as aproximações anteriores, nas quais sempre deve ser tomada uma decisão. Devido ao elevado custo computacional que será originado se sempre forem mantidas as indecisões. Desta forma conseguimos limitar o *fechamento* em cada iteração do algoritmo, mantendo somente um conjunto limitado de *k* disjunções.

Passo 7 - Regras de não inconsistência.

Caso seja necessário a realização de uma ordenação concreta, podemos verificar se a escolha será inconsistente e se temos alguma restrição disjuntiva pendente.

Nesta etapa será verificada a não consistência, que tem por objetivo comprovar se a decisão não irá causar problemas "futuros". Se a ordenação o_i

$\rightarrow o_j$ gera inconsistências "futuras", então deveremos trocar a decisão por $o_j \rightarrow o_i$, a menos que as regras detectam também problemas nesta alternativa, em cujo caso se procederá a realizar um processo de *backtracking*.

Supondo que tenhamos retificado a decisão $o_i \rightarrow o_j$ com as regras de não inconsistência, o processo pode ser realizado de forma mais eficiente se consideramos as indecisões pendentes. No caso de existir na rede de ordenações algumas sem decidir, será necessário então a realização de uma revisão das indecisões pendentes, já que será necessário tomar uma decisão, buscando reduzir o novo espaço de busca.

Passo 8 - Armazenamento de restrições não consideradas.

Na ultima etapa finalmente armazenamos as restrições associadas a decisões desejadas, assim, em caso de se detectarem inconsistências futuras, poderá ser realizado um *backtracking*, explorando as decisões não consideradas anteriormente.

Todo o processo é repetido até que fiquem restrições disjuntivas pendentes de seqüenciamento. Se em determinado momento não forem mais detectadas inconsistências, obtemos como resultado um conjunto mínimo de soluções.

Detecção de não disjunções

Dado um conjunto de restrições já identificadas na rede, pode ocorrer que, ao inserir uma nova restrição disjuntiva, entre dois pontos do tempo determinados, esta se converta em não disjuntiva, ao ser inconsistente uma das duas alternativas de seqüenciamento com alguma das restrições elementares já existentes na rede entre estes pontos. A detecção de não disjunções é importante devido qual o custe computacional de inserção de uma restrição não disjuntiva é menor que e das disjuntivas; além do mais, mantém uma única alternativa possível na rede, fazendo assim que o processo de busca seja mais eficiente. Para isso definimos novas regras que chamaremos *regras disjuntivas* (Etapa 4 do algoritmo *Fechamento_PSR*), que consistem no seguinte:

Suponhamos duas operações pendentes de seqüenciamento (o_i,o_j) *e* a correspondente restrição disjuntiva associada $(t_i\ c_{ij}\ t_j) \subseteq C$. Atualmente na rede existe uma restrição etiquetada entre dois pontos t_i e t_j, descrita por $(t_i\ lc'_{ij}\ t_j)$. Chamaremos c'_{ij} a restrição que se obtém a partir de lc'_{ij}, porém sem considerar suas etiquetas. A restrição c_{ij} pendente tem a forma, sendo *a e b* intervalos temporais, e representam a seqüência, respectivamente. Aplicando as regras (2) e (3) de forma que o algoritmo para detectar restrições não disjuntivas esta representado a seguir.

Como conseqüência deste processo obtemos um conjunto de restrições, que chamamos *ND*, que deveremos propagar antes que o resto das restrições disjuntivas. Se ao propagar o conjunto *ND* detectamos alguma inconsistência, então teremos que realizar o *backtracking*. No caso de que não haja nenhuma indecisão pendente terminaremos o processo, já que o problema não tem solução. Na realidade o que estamos fazendo ao eleger primeiro aquelas seqüências de ordenação que não admitem alternativas, é tentar detectar inconsistências o mais cedo possível. E assim dirigir a busca de forma mais eficiente.

Descrição do módulo de escalonamento

Módulo de escalonamento

Definição

Escalonamento de tarefas

Entradas

C = conjunto de restrições temporais

c_{ij} = restrições disjuntivas

T0 = início do intervalo

T = término do intervalo

hvalores = heurísticas de ordenação de valores

hvariáveis = heurísticas de ordenação de variáveis

Criar o conjunto de restrições temporais: C ={ $t_i\ c_{ij}\ t_j$ }; //PASSO 1

Selecionar e propagar as restrições não disjuntivas $c_{ij} \in$ C// PASSO 2

IF redundantes ≠ Ø THEN aplicar a regra de exclusão a C; END IF; //PASSO 3

IF detectamos uma inconsistência THEN erro; END IF;

WHILE C ≠Ø DO

Limitar o final do escalonamento c TO,T // PASSO 4

Detectar e propagar possíveis restrições não disjuntivas cij ∈C

IF detectamos uma inconsistência THEN backtracking; END IF;

IF hvariáveis ≠ Ø

THEN c_{ij} = heurísticas de ordenação de variáveis; //(local ou global) PASSO 5

IF NOT c_{ij} = seguinte restrição em ordem END IF;

Propagar c_{ij} ∈C //PASSO 6

IF detectamos uma inconsistência THEN backtracking; END IF;

IF hvalores ≠Ø

THEN c'_{ij} = aplicar heurística de ordenação de valores (local ou global)

IF NOT c'_{ij} = eleger uma ordem qualquer END IF;

IF c'_{ij} ≠Ø //PASSO 7

THEN

Propagar $c'_{ij} \in c_{ij}$

Verificar inconsistências com as restrições pendentes

Revisamos as indecisões pendentes

IF detectamos uma inconsistência THEN backtracking; END IF

IF NOT

IF $|c_{ij}|$> indecisões THEN escolher uma ordem qualquer END IF;

END IF

END IF

Guardar restrições desejadas //PASSSO 8

Devolver rede resultante

END Fechamento/PSR;

4.4 Conclusão

Neste capítulo o algoritmo de resolução proposto para resolver os problemas de restrições temporais, mostramos que podem ser tratados como restrições métrico disjuntivas. O algoritmo trabalha de forma iterativa, realizando sucessivos processos de fechamento/PSR, passos 4, 6 e 7 do módulo de fechamento, sobre as restrições fornecidas como entrada. O módulo de fechamento permite ainda trabalhar restrições com mais de uma disjunção, permitindo processar a cada vez mais de um valor possível.

O algoritmo pode executar backtracking se alguma das escolhas realizadas se mostrar inconsistente, com as condições e escolhas realizadas. Outra característica a ser destacada é que o algoritmo mantem as indecisões, o leva a um menor número de *backtracking* necessários para a obtenção da solução.

Finalmente o algoritmo proposto pode ser adaptado para lidar com restrições de recursos, seguindo basicamente o mesmo raciocínio implementado.

4.5 Exemplo de aplicação

Seja o seguinte exemplo, dado a necessidade de serem realizadas tarefas domesticas por um robô. Considerando as diversas restrições a serem consideradas, temporais (tais como limite de tempo e de realização de duas tarefas ao mesmo tempo), ou de recursos (não ser possível arrumar a cama e varrer o chão).

Neste caso o algoritmo escolhe do conjunto de tarefas a serem realizadas uma e verifica as restrições de tempo e de recursos, supondo que seja escolhida a arrumação de uma cama, será verificado pelo módulo de planejamento as restrições de recursos, tempo e precedência para a realização das taferas.

Na primeira etapa o módulo de planejamento verifica a consistência, caso não existam conflitos é realizada uma chamada ao módulo de escalonamento para verificar a consistência temporal das ações a serem realizadas.

Caso o modulo de escalonamento verifique uma inconsistência, o módulo de planejamento é novamente acionado para que seja escolhida nova tarefa.