Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

# Sistema de Escalonamento de Processos Baseado em Casos

**Ronaldo Camilo dos Santos**

Orientador: Prof. Dr. Juan Manuel Adán Coello
(Instituto de Informática/PUC-Campinas)
Co-Orientador: Prof. Dr. Maurício Ferreira Magalhães
(DCA/FEEC/UNICAMP)

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica

Este exemplar corresponde a redação final da tese defendida por RONALDO CAMILO DOS SANTOS e aprovada pela Comissão Julgada em 06 / 02 / 98

Orientador

Campinas, fevereiro de 1998

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Santos, Ronaldo Camilo dos
Sa59s Sistema de escalonamento de processos baseado em casos / Ronaldo Camilo dos Santos.--Campinas, SP: [s.n.], 1998.

Orientadores: Juan Manuel Adán Coello; Maurício Ferreira Magalhães.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Inteligência artificial. 2. Processamento eletrônico de dados em tempo real. 3. Analogia (Lógica). 4. Memória. 5. Raciocínio. 6. Grafos de ligação. 7. Semelhança (Geometria). I. Adán Coello, Juan Manuel. II. Magalhães, Maurício Ferreira. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

# Sistema de Escalonamento de Processos Baseado em Casos

"Sonho que se sonha só é
sonho só.
Sonho que se sonha junto é
realidade"

Raul Santos Seixas

Ao meu pai Adão Camilo dos Santos
À minha mãe Maria da Graça Santos
Às minhas irmãs Rosemeire, Rossenita e Regiane Camilo dos Santos

# Agradecimentos

À minha família, em especial aos meus pais e minhas irmãs, pelo estímulo durante o mestrado e em outros momentos igualmente importantes.

Aos professores Juan Manuel Adán (Instituto de Informática/PUC-Campinas) e Maurício Magalhães (DCA/FEEC/Unicamp) pelo tempo dedicado à orientação e co-orientação, pelo incentivo e pela oportunidade de realização deste trabalho.

Aos professores Maurício Figueiredo, Eleri Cardozo e Alice Tokarnia pelas contribuições à minha defesa de dissertação.

Ao pesquisador Alencar de Melo Jr., que colocou seu trabalho de mestrado e seus préstimos à disposição, colaborando para o andamento desta pesquisa.

Ao amigo Ilhas e à amiga Anna pelas inúmeras discussões e sugestões durante a realização deste trabalho.

Ao amigo Luís Otávio pelo incentivo e solidariedade.

A vocês, meus amigos, pela compreensão e pelo incentivo, nos momentos mais críticos desta e de outras jornadas.

À diretoria do Instituto de Informática da PUC-Campinas, que permitiu o acesso aos seus laboratórios e equipamentos, demonstrando o apoio sempre dispensado.

Às Bibliotecas da Área de Informática da PUC-Campinas (mestrado e graduação), da Área de Engenharia da Unicamp e do CPqD da Telebrás pelo apoio às pesquisas bibliográficas.

À equipe do LCA/FEEC/Unicamp pelo suporte técnico necessário à realização da parte experimental deste trabalho.

À CAPES pelo apoio financeiro fornecido durante a etapa de estudo e conclusão do mestrado.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Resumo

A necessidade de se ter tarefas processadas em uma planta de manufatura, clientes de banco aguardando para serem atendidos pelos caixas, aeronaves esperando autorização para pousar e unidades de programa para serem executadas em um computador têm em comum o problema de escalonamento que surge quando há necessidade de se escolher a ordem na qual as tarefas devem ser executadas e a atribuição de tarefas aos servidores para processamento. O problema de escalonar processos computacionais pode ser resolvido automaticamente por um algoritmo dedicado. Encontrar uma solução no menor tempo possível é um dos objetivos pretendidos.

A metodologia Raciocínio Baseado em Casos (RBC), que é uma abordagem para representação de conhecimento e utilização deste conhecimento para auxiliar na resolução de novos problemas, torna possível reduzir o tempo gasto para se encontrar uma solução à medida que os problemas ficam maiores. Este trabalho apresenta a proposta e implementação de um sistema baseado no método RBC para a resolução de problemas de escalonamento de processos, em sistemas de tempo real crítico estático e centralizado em um ambiente monoprocessado. Esses processos devem respeitar restrições de tempo de pronto, tempo de execução e prazo de término, além das relações de precedência.

O Sistema de Escalonamento de Processos Baseado em Casos (SEBC), implementado neste trabalho, possui como característica principal a idéia de simplificação de um problema através de substituições de seus subproblemas similares a problemas armazenados em uma base de casos e adaptação das soluções destes problemas na busca de uma solução para o problema apresentado. Os resultados obtidos pelo SEBC para os problemas estudados mostram que à medida que tratamos com problemas maiores, a reutilização de soluções de problemas passados pode reduzir substancialmente o tempo necessário para resolver esses problemas, ou detectar que os mesmos não têm solução.

# Abstract

Multiple jobs that should be processed in a manufacturing plant, bank customers waiting to be served by tellers, aircrafts waiting for landing clearances, and program tasks to be run on a computer have in common the scheduling problem that emerges whenever there is a choice concerning the order in which tasks can be performed and the assignment of tasks to servers for processing. The scheduling problem of computer processes can be automatically solved with a dedicated algorithm. One of the goals is to find a solution whenever one exists in a minimum time.

The Case-Based Reasoning methodology (CBR), an approach for knowledge representation and problem solving, makes it possible to reduce processing time to get a solution when the problems become large. This work implements and evaluates a CBR-based algorithm for solving the problem of process scheduling, in critical real time systems with a monoprocessor environment. These processes must respect restrictions of ready time, computation time and deadline, besides precedence relations.

The Case-Based Reasoning Real-Time Scheduler system (CBR-RTS), implemented in this work, holds, as main characteristic, the idea of simplification of a problem. This is made by substitutions of parts of the problem, that are in a case base, and adaptation of that problems aiming a solution for the presented problem. The experiments described in this work suggest that the CBR-RTS system can contribute to an expressive reduction in processing times required to schedule complex problems. In the tested examples, our algorithm performed better than the dedicated algorithm for large problems.

# 1

# Introdução

## 1.1 Motivação e Relevância

As pessoas, em diversas situações do dia a dia, deparam-se com problemas que vão da categoria mais simples à mais complexa. Mais do que um desafio, resolvê-los é, muitas vezes, uma questão de sobrevivência. Determinados problemas devem ser resolvidos com freqüência e considerando-se a sua classe, apesar de algumas variações existentes, geralmente o caminho que levou à solução de um problema pode ser utilizado para outros problemas.

Muitos destes problemas têm sido resolvidos automaticamente com o auxílio de algoritmos específicos, ou seja, dedicados. Encontrar uma solução no menor tempo possível é um dos objetivos pretendidos.

O problema de escalonamento representa um domínio interessante a ser abordado, pois é de muita utilidade em várias áreas. A necessidade de se ter tarefas processadas em uma planta de manufatura, clientes de banco aguardando para serem atendidos pelos caixas, aeronaves esperando autorização para pousar e unidades de programa para serem executadas em um computador paralelo ou distribuído têm em comum o problema de escalonamento que surge quando há necessidade de se escolher a ordem na qual as tarefas devem ser executadas e a atribuição de tarefas aos servidores para processamento.

Em geral, o problema de escalonamento engloba um conjunto de recursos e um conjunto de consumidores a serem atendidos por esses recursos de acordo com uma determinada política. A natureza dos consumidores e dos recursos, bem como as restrições sobre eles, afetam a escolha de uma política que gerencie a maneira como os consumidores têm acesso e utilizam os recursos a fim de otimizar alguma medida de desempenho. Por esta razão, um sistema de escalonamento reúne um conjunto de consumidores, um conjunto de recursos e uma política de escalonamento.

Um processo em um programa, uma tarefa em uma fábrica e um cliente em um banco são exemplos de consumidores. Um elemento de processamento em um sistema de computação, uma máquina em uma fábrica e um caixa em um banco são exemplos de recursos. O primeiro a chegar será o primeiro a ser servido (*First Come, First Served* - FCFS) é um exemplo de política de escalonamento, cujo desempenho varia com as circunstâncias. Enquanto a política FCFS é razoavelmente apropriada em um banco, ela pode não ser a melhor política para tarefas no chão de fábrica ou para processos em um sistema de computação
[El-Rewini e Ali, 1995].

Uma idéia interessante é armazenar, em uma base de casos, vários problemas já resolvidos e resgatar um ou mais deles com suas soluções para auxiliar na resolução de um novo problema. A simplificação deste problema, a partir da reutilização dos casos que são subpartes similares complementam esta idéia. Com este método, torna-se possível reduzir o tempo gasto para se encontrar uma solução. Adicionalmente, quando não existirem casos similares que levem a uma solução direta, pode-se passar um problema de menor complexidade a um algoritmo dedicado, favorecendo a resolução do novo problema proposto. Conseqüentemente, um novo par problema-solução pode incrementar a base de casos existente.

O método de se resolver problemas, baseado no conhecimento obtido a partir de resoluções anteriores, é formalmente conhecido, no universo da Inteligência Artificial, como Raciocínio Baseado em Casos (RBC). O método RBC é uma abordagem para representação de conhecimento e utilização deste conhecimento para auxiliar na resolução de novos problemas. Nesta abordagem, o conhecimento é representado como casos descrevendo experiências anteriores, as quais podem ser invocadas para auxiliar o raciocínio necessário para resolver um novo problema similar.

O RBC pode ser visto como um processo de raciocínio humano, uma representação de processos de raciocínio e um método de desenvolvimento de aplicações para a resolução de problemas. O RBC fornece uma metodologia para a construção de sistemas computacionais inteligentes e um modelo cognitivo humano. Psicólogos têm estudado a maneira como as pessoas usualmente resolvem seus problemas. Eles têm observado que, apesar de as pessoas serem hábeis em utilizar casos análogos resolvidos anteriormente, elas encontram certa dificuldade em relembrá-los. Para o computador, este processo de lembrança pode se tornar uma tarefa menos complicada quando bem definida [Kolodner, 1991].

O RBC é um paradigma de resolução de problemas e com ele um novo problema é resolvido buscando-se um caso similar passado e reutilizando-o quando da ocorrência de um novo problema. O método RBC apresenta também a característica de aprendizagem incremental, pois uma nova experiência é armazenada a cada vez que um problema é resolvido e torna-se imediatamente disponível para problemas futuros. A área de Raciocínio Baseado em Casos tem crescido rapidamente nos últimos anos, fato comprovado pelo aumento de publicações nas principais conferências, pelas ferramentas comerciais disponíveis e pelo sucesso das aplicações no uso diário [Aamodt e Plaza, 1996].

## 1.2 Contexto

Um sistema de Raciocínio Baseado em Casos (RBC) possui dois componentes principais: uma base de casos e um resolvedor de problemas. A base de casos contém casos, que são descrições de problemas anteriormente considerados e podem possuir ou não uma solução. O resolvedor de problemas é composto por dois módulos: um módulo de recuperação de casos e um módulo de inferência (raciocínio) sobre casos. Dado um novo problema, o módulo de recuperação identifica o caso mais apropriado ou os casos mais apropriados e apresenta o resultado desta recuperação ao módulo de inferência. A recuperação de casos possui duas fases: acesso ao caso e avaliação de similaridade. O módulo de inferência examina os casos recuperados e tenta resolver o novo problema, adaptando estes casos [Simoudis, 1992].

Segundo Cunningham e Smyth (1996), no momento, existe muito otimismo em relação à utilidade do método RBC para o desenvolvimento de sistemas baseados em conhecimento. O ponto que desperta este interesse é que casos podem representar soluções com boa qualidade, as quais podem vir a ser reutilizadas em novas situações, onde a reutilização de soluções pode ser vantajosa. Ele explorou este tipo de reutilização em problemas de escalonamento de tarefas numa situação na qual o tempo de pronto da tarefa é dependente da seqüência.

Conforme exposto na seção anterior, o método RBC é uma abordagem para representação de conhecimento e a utilização deste conhecimento para auxiliar na resolução de novos problemas. O domínio abordado neste trabalho é o escalonamento de processos computacionais. Portanto, o conhecimento representado é o problema de escalonamento com suas características próprias. A base inicial de conhecimento é fornecida por um algoritmo dedicado da área abordada. Neste trabalho, considera-se que o problema de escalonamento é composto de subproblemas e que ele pode ser resolvido a partir da resolução desses subproblemas individualmente. Reutiliza-se o problema, que seja mais similar ao novo subproblema apresentado. Existe um processo de raciocínio que, eventualmente, contribui para o aumento do conhecimento do sistema.

O problema a ser resolvido é o escalonamento de processos periódicos em um Sistema de Tempo Real Crítico estático e centralizado monoprocessado, cujo contexto motiva uma solução baseada em casos. Tipicamente, escalonamentos devem ser produzidos com determinada freqüência e o conjunto de processos a ser escalonado pode ser similar a um outro conjunto escalonado anteriormente [Cunningham e Smyth, 1996]. Em particular, o problema considerado neste trabalho possui as seguintes características: restrições de tempo, periodicidade e relações de precedência.

O trabalho está inserido neste contexto. É descrita a utilização do método RBC no desenvolvimento da aplicação proposta, suas características são discutidas e os resultados de simulações e avaliações de desempenho são apresentados.

## 1.3 Objetivos e Resultados

Este trabalho tem como objetivo principal estudar a utilização do método Raciocínio Baseado em Casos (RBC) para a resolução de problemas. Como domínio da aplicação, foi escolhido o problema de escalonamento de processos computacionais, isto é, o problema de escalonamento em sistemas de tempo real (STR) estático e centralizado.

Neste trabalho, considera-se que um novo problema é composto por subproblemas e que a sua solução pode ser encontrada através das soluções de seus subproblemas. A reutilização de problemas de escalonamento resolvidos anteriormente para solução de novos problemas envolve a manutenção de uma base de casos, a recuperação de um ou mais casos similares para um novo problema de escalonamento e a construção de um novo escalonamento a partir destes casos recuperados. O ciclo do raciocínio baseado em casos segundo Kolodner (1993) para se resolver um problema apresenta como passos principais: a recuperação de casos, a proposta de esboço de uma solução, a adaptação, o raciocínio avaliativo (justificativa e crítica), o teste avaliativo e a atualização da memória de casos.

No contexto da metodologia RBC e do problema de escalonamento de processos computacionais, este trabalho propõe o desenvolvimento de um sistema de raciocínio baseado em casos para resolução de tais problemas. O sistema é constituído por cinco módulos principais: representação de problemas, recuperação de casos (pares problema-solução), verificação de condições para a reutilização de soluções de problemas similares, substituição e aprendizagem. A representação de problemas e recuperação de casos no sistema proposto e implementado neste trabalho corresponde às fases de recuperação e proposta de um esboço de solução do ciclo RBC de Kolodner (1993). A verificação de condições para a reutilização de soluções de problemas similares e a substituição correspondem aos passos de adaptação, raciocínio avaliativo (justificativa e crítica) e teste avaliativo. O módulo de aprendizagem está relacionado à fase de aprendizagem do ciclo RBC.

O módulo de representação de problemas trata do armazenamento de informação, isto é, a construção da base de conhecimento. Este módulo é fundamental para um bom desempenho do sistema RBC, tanto em termos da qualidade da base como da maneira utilizada para representá-la. Inicialmente, alguns casos são fornecidos para a base e na resolução dos primeiros problemas o sistema irá mais aprender do que efetivamente resolver os problemas propostos.

A recuperação de casos é a função do segundo módulo e depende do tipo de representação utilizado, que determinará o mecanismo de busca e o tipo de avaliação de similaridade. A representação estruturada através de grafos acíclicos dirigidos mostra-se interessante para o problema, mas algumas dificuldades são encontradas em relação ao método para verificação de similaridade. No caso deste trabalho, com a utilização de grafos acíclicos para representação dos problemas de escalonamento, a utilização de algoritmo para detecção de isomorfismo de grafos e subgrafos (verificação da similaridade) é o ponto que

determina a eficiência em termos de tempo de processamento. Este algoritmo para detecção de isomorfismo de grafos e subgrafos se encerra em um problema do tipo *NP-Completo*.

O terceiro módulo tem por função selecionar (propor) a partir dos casos recuperados pelo passo anterior os casos que forem úteis para a resolução de subproblemas, verificando se as condições para a reutilização de soluções de problemas similares armazenados na base de casos são atendidas e se subproblemas podem ou não ser substituíveis.

O quarto módulo (substituição) trata a proposta de uma solução para o subproblema corrente. Neste trabalho, ela é implementada de uma maneira simples no sentido de que a solução do caso passado mais similar ao subproblema é apresentada como a solução para o subproblema do problema novo. Um subproblema possui um processo entrada e uma saída. Este subproblema, composto por alguns processos, é substituído por um único processo, cujo tempo de pronto é o tempo de pronto do processo de entrada, o tempo de execução é o tempo final menos o tempo inicial da solução do problema similar armazenado na base e o prazo de término é o maior prazo de término entre os processos do subproblema em questão. Quanto mais problemas, resolvidos anteriormente, houver, existe o potencial de mais subproblemas poderem ser substituídos.

O último módulo é a atualização, que permite efetuar a adição, à base de casos, de um problema simplificado, o qual não foi possível resolver com o conhecimento disponível até o momento, juntamente com a solução fornecida pelo algoritmo dedicado. Conforme novos problemas vão sendo apresentados para o sistema, a tendência é a base de conhecimento se tornar estável, isto é, os problemas devem ser resolvidos com o conhecimento já adquirido e o processo de aprendizagem passa a ser menos requisitado.

As simulações realizadas no trabalho mostram que o sistema de RBC aprende novos casos a partir de problemas resolvidos pelo algoritmo dedicado e, considerando-se a base de conhecimento existente, resolve determinados problemas de maneira mais rápida e consegue colocar à disposição do algoritmo dedicado um problema simplificado, que também possibilita encontrar uma solução em um menor tempo, mesmo não sendo através da base de casos existente até o momento. Portanto, ele é capaz de gerar uma base de conhecimentos que tende a ser útil para um número grande de problemas apresentados ao sistema.

## 1.4 Organização do Trabalho

Este primeiro capítulo apresentou o contexto dentro do qual o trabalho foi desenvolvido, seus aspectos relevantes, o resumo dos objetivos e principais resultados alcançados. O capítulo 2 faz uma caracterização da metodologia Raciocínio Baseado em Casos (RBC) e do escalonamento de processos em Sistemas de Tempo Real Crítico (STRC), que visa facilitar a explanação do trabalho realizado e a compreensão do texto. O contexto em que está inserido e os requisitos que devem ser analisados nesta proposta são determinados. Adicionalmente, uma revisão bibliográfica dos principais trabalhos disponíveis na literatura

sobre sistemas construídos de acordo com a metodologia RBC é apresentada. O capítulo 3 descreve o Sistema de Escalonamento Baseado em Casos, proposto neste trabalho, trata dos aspectos centrais da implementação do sistema e as suas características, além de detalhar como o sistema pode ser expandido para problemas com outras características. O capítulo 4 analisa os resultados de simulações conduzidas para avaliar o sistema desenvolvido na resolução de alguns problemas escolhidos de escalonamento de processos em STRC. O capítulo 5 conclui o trabalho, fazendo uma retrospectiva de suas contribuições e apresentando sugestões para pesquisas futuras. O anexo A apresenta os casos (pares problema-solução) que formam a base de conhecimento inicial usada na análise preliminar do desempenho do sistema proposto e implementado.

# 2

# Caracterizando RBC e Escalonamento

Este capítulo faz uma descrição da metodologia Raciocínio Baseado em Casos (RBC) e apresenta o ciclo do RBC e suas partes segundo Kolodner (1993). Alguns sistemas baseados em casos são apresentados e é feita uma discussão sobre alguns trabalhos correlatos envolvendo a aplicação de RBC a escalonamento. Adicionalmente, o problema de escalonamento em Sistemas de Tempo Real Crítico (STRC) é descrito de forma geral, tomando por base a dissertação de mestrado de Melo Jr. (1993). É mostrado como o problema de escalonamento é resolvido segundo o algoritmo de Xu e Parnas (1990), através do Escalonador de Processos Periódicos e Esporádicos (EPPE), implementado por Melo Jr. (1993). Feito isso, são descritas as características consideradas neste trabalho para o problema abordado. Finalmente, a utilização da metodologia RBC para resolução do problema de escalonamento de processos é introduzida, em particular, naqueles casos onde determinados problemas possam ser resolvidos em menor tempo que com o algoritmo dedicado.

## 2.1 Raciocínio Baseado em Casos

### *2.1.1 Definição de Raciocínio Baseado em Casos*

Segundo Schank (1982), a Inteligência Artificial (IA), na sua forma mais simples, pode ser vista como o estudo que visa a capacitar os computadores para resolverem problemas de uma maneira similar à do ser humano. O raciocínio humano pode ser visto como sendo baseado numa coleção de experiências e casos armazenados, em uma memória dinâmica. Com certa freqüência, os seres humanos resolvem problemas, coletando e adaptando o conhecimento obtido a partir de soluções de problemas similares aos atuais encontradas anteriormente, que estejam armazenadas nesta memória dinâmica. Quando as pessoas se deparam com um novo tipo de problema, elas buscam o conhecimento requerido para resolver o problema através de algum método tradicional. Em seguida, armazenam esta informação, recentemente adquirida, na memória dinâmica para ser utilizada na resolução de um problema futuro. Este método de resolução de problemas, baseado no conhecimento obtido a partir de

soluções anteriores, é formalmente conhecido no universo de IA como Raciocínio Baseado em Casos (RBC) [Kolodner, 1993].

Segundo Kolodner (1993), RBC é um modelo de raciocínio, que envolve o entendimento de situações, a resolução de problemas e a aprendizagem, integrando esses aspectos a processos de memória. As seguintes premissas são subjacentes ao modelo:

- ✧ referência a casos passados é vantajosa quando se trata situações que se repetem. A referência a situações prévias similares é, freqüentemente, necessária para tratar novas situações complexas. Por isso, relembrar um caso para usá-lo mais tarde na resolução de um problema (e integrar este caso com o que já é conhecido) é um processo de aprendizagem necessário;

- ✧ como descrições de problemas são geralmente incompletas, o entendimento ou interpretação do problema é um pré-requisito necessário para raciocinar. Um módulo de inferência (raciocínio) baseado em casos não pode relembrar um caso relevante a menos que ele entenda a situação que está tratando. Por outro lado, conforme a resolução do problema progride, um módulo de inferência pode obter uma melhor compreensão de uma situação, permitindo que mais casos relevantes sejam relembrados do que antes. Isto indica que compreender ou interpretar uma situação é uma parte necessária do ciclo do raciocínio e um pré-requisito para a resolução do problema e um co-requisito durante a resolução. Mas a necessidade de se compreender o problema não é específica ao raciocínio baseado em casos. Qualquer forma de raciocínio requer que uma situação seja elaborada detalhadamente e representada com clareza e com vocabulário apropriado para permitir que o módulo de inferência reconheça o conhecimento que ele precisa (conhecimento genérico ou casos) para raciocinar sobre ele;

- ✧ como casos antigos nem sempre são idênticos aos novos, usualmente é necessário adaptar uma solução conhecida para satisfazer às demandas de uma nova situação. A adaptação compensa as diferenças entre uma situação passada e uma nova;

- ✧ a aprendizagem ocorre como uma conseqüência natural do processo de raciocínio. Se um novo procedimento é derivado no decorrer da resolução de um problema complexo e tudo ocorre bem nesta sua execução, então, como conseqüência, um novo procedimento é aprendido para tratar esta nova classe de situações. O procedimento é incorporado ao caso em questão, e o caso é indexado na memória de maneira que seja recuperável quando seu procedimento puder ser usado com vantagem. Se for desvantajoso e problemas forem encontrados utilizando este novo caso em uma nova situação, o módulo de inferência é advertido de que o procedimento no caso é falho ou que seus índices (representando sua faixa de aplicabilidade) não são acurados. É feita uma tentativa para analisar os resultados da nova situação e contornar seus problemas. A nova situação, armazenada na biblioteca de casos, engloba um refinamento ou uma modificação do conhecimento do raciocínio encontrado no caso original. Seus índices designam quando ele é útil e índices associados aos casos passados são refinados, baseado

nesta análise, de maneira que ele é recuperado somente quando se sabe que seu procedimento é apropriado. Este processo de aprendizagem incremental resulta na aprendizagem de novos procedimentos, no seu refinamento e na aprendizagem de quando cada procedimento pode ser apropriadamente utilizado, e

- realimentação e análise da realimentação são partes necessárias do ciclo de raciocínio/aprendizado. Sem os processos de avaliação baseados na realimentação, a aprendizagem poderia não acontecer e referências a experiências prévias durante o raciocínio não seriam possíveis.

Estas premissas sugerem que a qualidade de um módulo de inferência (raciocínio) baseado em casos depende de cinco aspectos [Kolodner, 1993]:

1. as experiências que teve;

2. a sua habilidade para entender novas situações em termos de experiências conhecidas. Isto pressupõe relembrar experiências prévias e interpretar a nova situação em termos das experiências relembradas;

3. a sua capacidade de adaptar soluções;

4. a sua capacidade de avaliar e reparar soluções, baseado em resultados de casos similares, realimentação ou simulações e

5. a sua habilidade para integrar apropriadamente novas experiências à sua memória.

### *2.1.2 Definição de Caso*

Um caso é uma peça de conhecimento contextualizado, representando uma experiência que ensina uma lição fundamental para alcançar os objetivos do módulo de inferência (raciocínio). Existe muita diversidade no conteúdo e forma dos casos, mas eles, geralmente, possuem as seguintes características [Kolodner, 1993]:

1. casos contêm conhecimento útil para a resolução do problema que se tem em mãos. Este conhecimento está no nível operacional e ligado a um contexto específico, isto é, o conhecimento está pronto para ser aplicado;

2. o tamanho e a estrutura de um caso variam bastante. Uma instância de resolução de problema pode estar distribuída em um determinado número de casos ou armazenada em um único caso, e

3. somente o conhecimento de uma experiência que encapsula algo sobre a resolução de um problema deve ser armazenado como um caso, isto é, o conhecimento de um caso deve tornar mais fácil a resolução de um problema futuro.

### *2.1.3 O Processo de Raciocínio Baseado em Casos*

A Figura 1 apresenta uma ilustração do ciclo do Raciocínio Baseado em Casos (RBC), representando o processo de interpretação e assimilação de um novo conhecimento. Os casos podem servir a dois tipos de propósitos quando eles são relembrados:

- ✧ casos fornecem sugestões de soluções para problemas ou
- ✧ um contexto para compreensão ou avaliação de uma situação

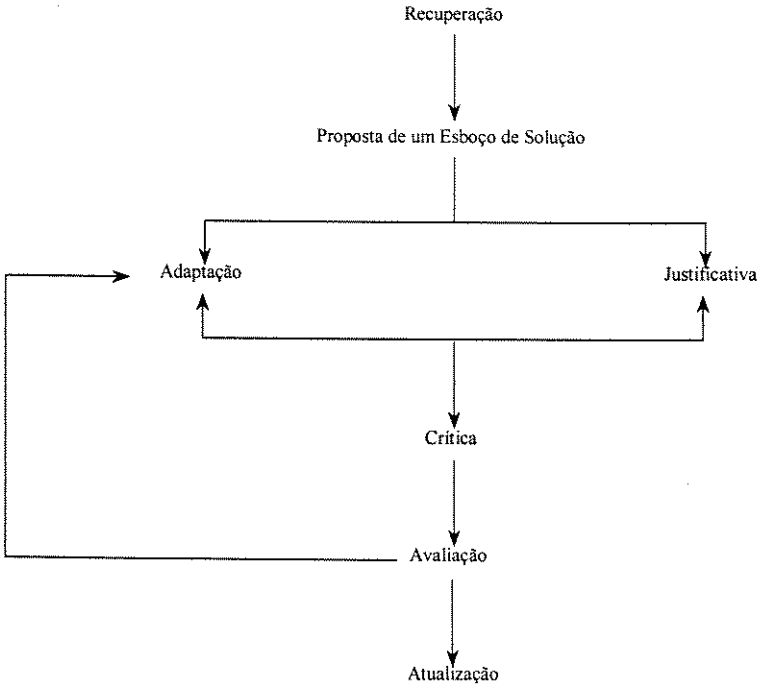



***Figura 1*** *Ciclo do Raciocínio Baseado em Casos [Kolodner, 1993]*

Como fornecedores de sugestão, os casos colocam à disposição esboços de soluções que são adaptadas para se ajustarem a uma nova situação. Como fornecedores de contexto, os casos apresentam uma evidência concreta a favor ou contra alguma solução ou interpretação que pode guiar procedimentos de interpretação ou avaliação. RBC é, por esta razão, um processo de relembrar e adaptar ou relembrar e comparar. As pessoas utilizam casos para ajudá-las a compreender e avaliar situações e a resolver problemas. Não se pode resolver um problema quando não se compreende a situação que o envolve; freqüentemente, necessita-se resolver novos problemas afim de compreender bem as implicações de uma situação. As soluções são avaliadas, projetando seus resultados, utilizando métodos de avaliação e pode ser necessário resolver novos problemas no decorrer da avaliação. Existem dois estilos diferentes

de RBC. Em situações de resolução de problema, tende-se a enfatizar o uso de casos para propor soluções; em situações de interpretação, tende-se a utilizar casos para gerar críticas e justificativas.

Esta discussão indica os processos primários requeridos para o RBC. Primeiramente, casos similares devem ser recuperados para facilitar o raciocínio. Por este motivo, a recuperação de casos é um processo primário, assim como o seu processo componente, casamento parcial e seu processo adjunto, armazenamento de casos (também chamado de atualização de memória). Afim de garantir que soluções pobres não sejam colocadas juntamente com as boas soluções, os dois estilos de RBC avaliam suas soluções.

Cada um dos estilos de raciocínio baseado em casos requer que um raciocínio diferente seja feito a cada vez que os casos são recuperados. No RBC para resolução de problemas, um esboço de solução para o novo problema é proposto, extraindo a solução de algum caso recuperado. Esta ação é seguida pela adaptação, o processo de ajustar uma solução passada a uma nova situação, e crítica, o processo de criticar a nova solução antes de utilizá-la efetivamente. No RBC para interpretação, um esboço de interpretação ou resultado desejado é proposto, algumas vezes baseado em casos recuperados, outras vezes imposta (por exemplo, quando o cliente de um advogado requer um determinado resultado). Isto é seguido pela justificativa, o processo de criar um argumento para a solução proposta, feito por um processo de comparar e contrastar a nova situação com casos prévios, buscando similaridades entre a nova situação e outras que justifiquem o resultado desejado e diferenças que impliquem que outros fatores devam ser levados em consideração. Algumas vezes a justificativa é seguida por um passo de crítica no qual situações hipotéticas são geradas e a solução proposta aplicada a eles a fim de testar a solução.

Estes passos e estilos são, num certo sentido, recursivos. Os passos de criticar e adaptar, por exemplo, freqüentemente requerem que novos casos sejam recuperados. Existem também várias malhas de repetição no processo. A crítica pode levar a uma adaptação adicional, assim como a avaliação. E quando o raciocínio não está progredindo bem, utilizando um determinado caso, pode ser necessário reiniciar todo o processo com um novo caso escolhido.

A seguir, são apresentados os passos de recuperação de casos, proposta de um esboço de solução, adaptação, avaliação do raciocínio (justificativa e crítica), teste de avaliação e atualização de memória. Cada passo é seguido de um resumo e alguns de seus aspectos [Kolodner, 1993].

### *Recuperação de Casos*

Relembrar é o processo de recuperar um caso ou um conjunto de casos da memória. Em geral, consiste de dois passos:

1. relembrar "bons" casos, isto é, aqueles que têm o potencial de fazer previsões relevantes sobre o novo caso e

2. selecionar os casos mais promissores, gerados no passo anterior.

Esta etapa do processo do RBC apresenta alguns problemas, a saber:

- ✧ deve-se capacitar o computador com um maneira de discernir qual caso é aplicável a uma nova situação. Este problema é o casamento (*matching*), ou avaliação de similaridade (*similarity-assessment*). Às vezes a semelhança entre casos não é evidente (por exemplo, estratégia em jogos competitivos). Estratégias no futebol e no xadrez têm muitos aspectos em comum, mas as características concretas dos jogos são dissimilares. Um é jogado num tabuleiro, o outro em um campo; um possui peças, o outro envolve pessoas; um possui times competindo entre si, o outro uma dupla disputando a partida. Contudo um especialista em xadrez poderia planejar uma jogada para ser aplicada em um jogo de futebol. Uma maneira de tratar este problema seria usar mais do que apenas uma representação dos aspectos físicos de um caso para comparação. Casos devem ser comparados também em níveis de representação abstrata;

- ✧ o vocabulário de indexação. Das maneiras abstratas de representar casos, quais são as mais apropriadas para usar em comparações? (particularmente para detectar semelhanças não evidentes);

- ✧ a avaliação da situação. Às vezes sabe-se tão pouco a respeito de uma nova situação que há pouca base para compará-la com outras. A questão aqui é determinar uma maneira de elaborar descrições de situações e gerar características derivadas para casos e

- ✧ o algoritmo de recuperação. Como fazer buscas em bases de casos eficientemente?

Esses problemas somados caracterizam o problema da indexação, isto é, como associar índices aos casos que determinem sob que condições cada caso pode ser usado para fazer inferências úteis.

***Propondo um esboço de solução***

Neste passo, porções relevantes dos casos selecionados durante a recuperação são extraídas para formar um esboço de solução para o novo caso. Na resolução de problema, este passo normalmente envolve selecionar a solução do problema passado, ou parte dela, como um esboço de solução para o caso em questão. Surgem vários aspectos durante a construção de um esboço de solução. O primeiro aspecto é a questão de como porções apropriadas de um caso passado podem ser selecionadas.

- ✧ como selecionar partes apropriadas de um caso conhecido?

1. focalizar a atenção na parte do caso que foi relevante para alcançar o objetivo atual no caso passado.

2. a estrutura interna do caso passado, especialmente as dependências entre diferentes partes do caso, sinalizam como expandir o foco do módulo de inferência (raciocínio) em direções relevantes. Portanto, as características do caso que levaram à seleção da solução ou classificação devem também ser focalizadas.

- ✧ geralmente, adaptações relativamente simples e automáticas podem ser feitas em soluções antigas antes de passar para as etapas de adaptação ou justificação. Por exemplo, em mediações trabalhistas, ajustes de salários e outros benefícios a partir do custo de vida.

- ✧ no raciocínio interpretativo, quando todas as alternativas não estão conectadas, a escolha inicial pode afetar muito a precisão da interpretação resultante.

***Adaptação***

Na resolução de problemas baseando-se em casos, soluções são usadas como inspiração para resolver novos problemas. Situações novas raramente casam exatamente com as passadas. Por isso, soluções antigas devem ser modificadas para se ajustarem às novas situações. Neste passo, chamado de adaptação, a solução proposta é adaptada para se ajustar à nova situação. Existem dois passos principais que constituem a adaptação de um esboço de solução:

1. determinar o que precisa ser adaptado e

2. fazer a adaptação.

Os aspectos da adaptação surgem destes dois passos. Inicia-se considerando a adaptação em si. Para qualquer domínio ou tarefa particulares, pode-se apresentar um conjunto de estratégias ou heurísticas de adaptação. Pode-se implementá-las e criar um sistema que as contemple. Devem ser observadas questões importantes:

- ✧ há um conjunto geral de estratégias de adaptação, válidas para qualquer domínio, a partir do qual se possa começar e que forneça diretrizes para definir estratégias de adaptação especializadas? Ex. remover um componente secundário: um componente secundário de um item pode ser removido (apagado) se não desempenha função necessária. Para cada tipo de estratégia de adaptação, deve-se determinar qual é o conhecimento necessário à sua aplicação e

- ✧ Elaborar metodologias para avaliar que partes de uma solução conhecida precisam ser adaptadas para aplicar ao novo problema. Por exemplo, verificar inconsistências entre soluções conhecidas e novas necessidades, e, baseado nisso, determinar o que deve ser adaptado.

***Raciocínio avaliativo: justificativa e crítica***

Nestes passos, uma solução ou interpretação é justificada, geralmente antes de ser aplicada. Quando todo o conhecimento necessário para avaliação é conhecido, pode-se pensar neste passo como um passo de validação. Porém, em muitas situações, existem muitos aspectos desconhecidos que impossibilitam a validação de uma solução. Os passos são os seguintes:

- ✧ soluções podem ser criticadas comparando-as e contrastando-as com soluções similares. Este passo requer uma chamada recursiva a processos de memória para recuperar casos com soluções similares;
- ✧ podem ser propostas situações hipotéticas para testar a robustez da solução;
- ✧ podem ser feitas simulações e verificar os resultados;
- ✧ pode ser necessário recuperar casos adicionais, e proceder a uma adaptação adicional, neste caso denominada reparação; e
- ✧ aspectos principais: estratégias para avaliação usando casos; estratégias para recuperação e uso de casos para interpretação, avaliação e justificativa; geração de hipóteses e estratégias para usá-las

***Teste avaliativo***

Os resultados da aplicação da solução proposta são obtidos e analisados. Uma realimentação sobre os acontecimentos durante a utilização da solução ou sobre o resultado desta utilização é obtida e analisada. Se os resultados estiverem conforme esperado, não é necessária uma análise mais avançada neste passo, mas se forem diferentes do esperado, torna-se necessária uma explicação dos resultados anômalos. Apresenta como características os seguintes itens:

- ✧ se os resultados forem diferentes do esperado deve-se procurar explicar o que causou a anomalia e determinar o que poderia ter sido feito para evitá-la;
- ✧ é um dos passos mais importantes para um módulo de inferência (raciocínio) RBC, pois permite que ele avalie o efeito das suas decisões no mundo real, possibilitando-lhe aprender;
- ✧ permite antecipar e evitar erros, e identificar possibilidades não notadas anteriormente que podem ser úteis no futuro; e
- ✧ pode indicar a necessidade de adaptação ou reparação adicional da solução proposta

***Atualização da memória de casos***

O novo caso é armazenado apropriadamente na memória para uso futuro.

- ✧ o aspecto mais importante neste passo é a escolha do modo de indexar o caso na memória (o módulo de inferência deve ser capaz de antecipar a importância do caso em raciocínios futuros).

### *2.1.4 Vantagens e Desvantagens de RBC*

O raciocínio baseado em casos apresenta muitas vantagens para um módulo de inferência (raciocínio).

- ✧ RBC permite propor soluções rapidamente, evitando o tempo necessário para derivá-las a partir do início.

Um médico relembrando um diagnóstico ou tratamento passado exemplifica este benefício. Apesar de um módulo de inferência baseado em casos ter que avaliar soluções propostas, como qualquer racionador faz, ele sai na frente na resolução de problemas, pois pode gerar propostas facilmente. Existe uma considerável vantagem em não se ter que refazer computações e inferências. Esta vantagem é útil para quase todas as tarefas de raciocínio, incluindo resolução de problemas, planejamento e diagnóstico. De fato, a avaliação do CASEY [Koton, 1988] mostra um incremento de velocidade de duas ordens de magnitude quando um problema foi visto no passado.

- ✧ RBC permite propor soluções em domínios que não são completamente entendidos pelo módulo de inferência.

Muitos domínios são impossíveis de se compreender completamente, geralmente porque muitos aspectos dependem da imprevisibilidade do comportamento humano (a economia, por exemplo). Outros, ninguém compreende ainda (por exemplo, como alguns medicamentos e doenças se manifestam). Outras vezes, simplesmente as pessoas se encontram em situações que não compreendem bem, mas nas quais devem agir de qualquer maneira, por exemplo, na escolha dos alunos de graduação a serem aceitos em um programa de mestrado. RBC permite que se faça previsões baseadas naquilo que funcionou no passado sem ter uma compreensão completa.

- ✧ RBC fornece elementos para avaliar soluções quando não há métodos algoritmicos disponíveis para avaliação

Utilizar casos para auxiliar na avaliação é particularmente útil quando há muitos aspectos desconhecidos, fazendo qualquer outro tipo de avaliação impossível ou difícil. Soluções são avaliadas no contexto de situações similares prévias. Novamente, o módulo de inferência faz esta avaliação baseado naquilo que funcionou no passado.

- ✧ casos são úteis na interpretação de conceitos/situações em aberto e mal definidos

É muito utilizado por advogados, mas é também importante para situações do dia-a-dia. O desempenho do PROTOS [Bareiss, 1989; Porter et al, 1990] em classificar disfunções auditivas quando se conhece pouca informação mostra que uma metodologia baseada em casos para interpretação pode ser mais acurada do que métodos tradicionais baseado em condições necessárias e suficiente quando classificações são mal definidas.

- ✧ relembrar experiências é particularmente útil para trazer à tona problemas que ocorreram no passado, alertando para a necessidade de ações que impeçam a repetição de equívocos passados e

- ✧ casos ajudam a focalizar a atenção em elementos importantes de um problema. O que foi importante para um sucesso ou fracasso numa situação passada tende a sê-lo numa nova situação

Evidentemente, existem também desvantagens em se raciocinar a partir de casos:

- ✧ Um módulo de inferência (raciocínio) RBC pode ser tentado a usar casos passados cegamente, confiando em experiências prévias sem validá-las na nova situação;

- ✧ casos podem direcionar excessivamente um módulo de inferência na solução de um novo problema e

- ✧ freqüentemente, pessoas, especialmente novatos, não se lembram dos casos mais apropriados quando estão raciocinando [Gick e Holyoak, 1980; Gentner, 1989]

### *2.1.5 Modelo cognitivo e metodologia para construir sistemas*

Há muitas evidências de que as pessoas utilizam RBC como parte do seu processo de raciocínio. Uma destas evidências é experimental. Ross (1986, 1989), por exemplo, mostrou que pessoas aprendendo uma nova habilidade geralmente se referem a problemas prévios para refrescar suas memórias em como realizar uma tarefa. Kolodner (1993) conduziu uma pesquisa que mostra que mecânicos iniciantes e experientes utilizam suas próprias experiências e as de outras pessoas para ajudá-los na geração de hipóteses sobre o que está errado com o carro, reconhecer problemas (e.g., um instrumento de teste que não funciona), e relembrar como aplicar estas experiências para diferentes diagnósticos [Lancaster e Kolodner, 1987, 1988]. Outra pesquisa realizada por Kolodner (1993) mostra que médicos usam muito casos prévios para gerar hipóteses sobre o que está errado com um paciente, para ajudá-los a interpretar resultados de testes e selecionar terapias quando várias estão disponíveis e nenhuma é compreendida muito bem. Pesquisadores na GTE descobriram que o raciocínio de engenheiros sobre o que poderia sair errado com redes de comutação usavam casos extensivamente [Kopeikina et al, 1988]. Psicólogos têm descoberto que pessoas se sentem

confortáveis em utilizar casos para tomar decisões [Ross, 1986, 1989; Klein e Calderwood, 1988; Read e Cesa, 1990], mas nem sempre relembram os casos certos [Holyoak, 1985; Gentner, 1989]. Para aliviar este problema, o computador pode ser utilizado como uma ferramenta de recuperação para auxiliar as memórias das pessoas.

Como um método para construção de sistemas de raciocínio inteligentes, o RBC é atraente porque parece ser relativamente simples e natural. Várias pessoas construindo sistemas especialistas que sabem como raciocinar utilizando casos têm achado mais fácil construir sistemas especialistas baseados em casos do que aqueles tradicionais [Barletta e Hennessy, 1989; Goodman, 1989; Simoudis, 1992]. Um grande problema em raciocínio em domínios de especialistas é o alto grau de incerteza e incompleteza do conhecimento envolvido. Um módulo de inferência (raciocínio) RBC trata esses problemas baseando-se naquilo que funcionou no passado. Sistemas baseados em casos resolvem problemas de maneira mais eficiente que os sistemas dedicados tradicionais, que, para encontrarem uma solução, partem sempre do início sem poderem aproveitar passos já conhecidos [Koton, 1988, 1989a].

Há uma grande gama de sistemas RBC que se pode construir, indo dos totalmente automáticos, num extremo, aos que se restringem à recuperação de casos, no outro extremo. Sistemas totalmente automáticos são aqueles que resolvem problemas sozinhos e têm algum meio de interagir com o mundo para receber realimentação relativa às decisões tomadas. Sistemas apenas de recuperação interagem com uma pessoa para resolver um problema. Eles atuam de maneira a aumentar a memória de uma pessoa, fornecendo casos que a pessoa pode não conhecer. A pessoa é responsável por tomar as decisões (raciocinar) a partir dos casos.

### *2.1.6 O Potencial do RBC: Alguns Sistemas*

São apresentados resumidamente nesta seção dez sistemas baseados em casos citados em [Riesbeck e Schank, 1989].

O sistema **IPP** (*Integrated Partial Parser*) [Lebowitz, 1980] é um modelo de organização de memória e compreensão de texto dirigido por memória, usando MOPs (*Memory Organisation Packets*), modelo de memória dinâmica [Schank, 1982]. O IPP lê textos sobre atividades terroristas, armazena suas interpretações na memória, faz generalizações e utiliza estas generalizações para guiar a compreensão de histórias futuras. Elementos chaves do IPP são seu modelo de memória, suas regras para formar generalizações e o seu uso da memória para guiar o *parsing*.

O **CYRUS** [Kolodner, 1984] também é um sistema de compreensão de histórias baseado em MOP, mas com o foco em como a memória é utilizada para responder a questões depois da compreensão. CYRUS lê histórias sobre viagens diplomáticas, armazena suas interpretações, faz generalizações, e responde a questões, tais como "Quando você encontrou Begin pela última vez?" e "Você já encontrou a senhora Begin alguma vez?". A segunda questão é importante, pois não é plausível que CYRUS possa ou (a) olhar cada episódio na

memória a fim de responder uma questão, ou (b) indexar cada episódio previamente sob cada questão que pudesse ser feita sobre ele. Portanto, para responder a uma questão como "Você já encontrou a senhora Begin alguma vez?" CYRUS gera subquestões como "Quando eu poderia encontrar a esposa de um diplomata? (em um jantar oficial)", "Quando eu poderia ir a jantar oficial com Begin? (em uma visita diplomática a Israel)", etc. Tais subquestões são geradas até que uma memória seja recuperada com uma resposta, ou não haja mais elaborações possíveis.

O sistema **JUDGE** [Bain, 1986] atua no domínio de sentenças criminais. Ele modela um juiz que está determinando sentenças para pessoas condenadas por crimes. A entrada é uma descrição do caso, incluindo a acusação, os fatos que ocorreram e os estatutos legais que contemplam crimes desta natureza, por exemplo, tempo permitido de reclusão e as condições de livramento condicional.

A biblioteca de casos contém crimes prévios e as sentenças determinadas para cada um deles. JUDGE inicia com uma biblioteca de casos vazia e um conjunto vasto de heurísticas para a determinação de sentenças quando não houver casos que possam ser aplicados a uma nova situação. Ele em seguida constrói uma base de casos, trilhando suas próprias decisões passadas.

Este domínio não envolve realimentação. Uma sentença dentro de limites legais não pode ser considerada errada. Isto o torna um problema de avaliação subjetiva, similar a problemas de avaliação de riscos, etc. Em tal domínio, no qual certo e errado são basicamente impossíveis de se determinar, é importante ser consistente. O sistema JUDGE usa a biblioteca de casos para manter um padrão de sentenciamento constante. Um novo crime é primeiro interpretado, avaliando os fatos por seriedade, intencionalidade, justificativa, etc, guiado por casos similares recuperados da biblioteca. Finalmente, a sentença armazenada no caso recuperado é adaptada ao crime em questão, tornando a sentença mais ou menos severa.

O sistema **CHEF** [Hammond, 1989] atua no domínio da culinária. Ele gera novas receitas, adaptando receitas passadas. Este é um domínio de projeto, no qual um objeto tem de ser construído para satisfazer a vários objetivos simultaneamente. Arquitetura, programação e planejamento são outros exemplos de domínios de projeto.

CHEF inicia com uma biblioteca de aproximadamente vinte receitas. A entrada para o CHEF é uma lista de objetivos, tal como "dê-me um prato quente usando o método *stir-fry*, com galinha e brócoli". A saída é uma receita. O usuário do sistema CHEF em seguida avalia os resultados. Se houver problemas, o usuário submete um relatório de falhas para o CHEF. CHEF repara a receita e também modifica sua biblioteca de casos para afastar a possibilidade de se cometer o mesmo erro em situações similares no futuro.

CHEF difere do JUDGE no sentido de que ele aprende a partir de suas falhas e seu processo de adaptação é mais complexo. Por outro lado, a interpretação dos fatos em JUDGE é mais complexa do que a análise dos objetivos do CHEF.

O sistema **COACH** [Collins, 1987] atua no domínio do futebol . Ele gera novas jogadas de futebol, melhorando jogadas passadas. Este é um outro exemplo de domínio de projeto. O foco da pesquisa em COACH tem sido na depuração e reparação de planejamentos armazenados. Por esta razão, COACH possui poucas jogadas na sua biblioteca de casos, mas ele possui um número de estratégias para modificar jogadas para formar novas possibilidades. Qual modificação usar depende da natureza da falha encontrada na jogada existente.

O sistema **MEDIATOR** [Simpson, 1985] atua no domínio de resolução de disputa. Dado um conflito de objetivos entre várias partes, ele propõe alternativas possíveis. Se uma proposta falhar em satisfazer todas as partes envolvidas, MEDIATOR gera uma proposta nova. Ele também armazena um registro da falha da primeira proposta para ajudá-lo a prever e afastar tais falhas no futuro.

**HYPO** [Rissland e Ashley, 1986; Ashley, 1987] implementa um módulo de inferência (raciocínio) no domínio jurídico baseado em casos da área de lei de patentes. Dada uma descrição de um caso envolvendo alguma reclamação sobre violação de direitos, HYPO usa sua base de casos precedentes para gerar argumentos plausíveis para a acusação ou defesa. Por exemplo, dada uma descrição de um caso, tal como a venda de segredos industriais para um concorrente e o objetivo de argüição para a defesa, HYPO procura casos na memória que sejam similares ao caso apresentado e que foram julgados em favor da defesa. HYPO em seguida procura maneiras para reduzir aparentes diferenças, se existirem, entre o caso dado e os casos recuperados com sucesso.

O sistema **PLEXUS** [Alterman, 1986] é um planejador que adapta planejamentos passados a novas situações. O PLEXUS tem sido descrito através de um exemplo envolvendo a adaptação do planejamento para se deslocar no sistema de metrô de São Francisco para um planejamento utilizando o sistema de Nova Iorque. Apesar de o PLEXUS parecer não ter uma base significativa, ele é interessante em função dos mecanismos que utiliza para a adaptação dos planejamentos passados.

Assim como o MEDIATOR, o sistema **PERSUADER** [Sycara, 1987] propõe resoluções para situações de disputa. Como o HYPO, o PERSUADER também usa sua base de casos precedentes para gerar argumentos plausíveis contra ou a favor. O domínio do PERSUADER é negociações trabalhistas. Ele cria e argumenta contratos de trabalho. A entrada é uma descrição de uma disputa entre gerência e trabalhador sobre algum pacote de benefícios. PERSUADER cria um pacote que é um meio termo baseado nos objetivos dos participantes da discussão e os padrões da indústria. Ele faz isto adaptando contratos que já foram utilizados em companhias similares. Ao contrário de muitos módulos de inferência baseados em casos desenvolvidos, PERSUADER também possui um planejador reserva, capaz de gerar novos contratos quando não existir algum que possa ser encontrado ou adaptado.

O sistema **CASEY** [Koton, 1988] diagnostica doenças cardíacas. Ele considera uma descrição dos sintomas de um paciente e produz um modelo causal explicando possíveis desordens que possam levar àqueles sintomas. CASEY se situa no topo de um sistema de diagnóstico mais completo baseado em modelos, que é a fonte da biblioteca inicial de

diagnósticos. Quando novos casos são apresentados, CASEY procura casos de pacientes com sintomas similares, mas não necessariamente idênticos. Se encontrar um bom casamento, CASEY em seguida tenta adaptar o diagnóstico recuperado, levando em consideração diferenças em sintomas entre os casos passados e novos. Diagnósticos adaptados pelo CASEY assemelham-se muito com aqueles construídos pelo sistema completo baseado em modelo, mas são construídos muito mais eficientemente.

## 2.2 Aplicações de RBC a Escalonamento

Duas abordagens distintas para escalonamento baseado em casos podem ser identificadas na literatura [Cunningham e Smyth, 1996]. A primeira é a usada no SMARTplan de Koton (1989b). Casos são usados para propor escalonamentos preliminares, os quais são, em seguida, adaptados (refinados e reparados) para satisfazer os requisitos do escalonamento em questão. A segunda abordagem não utiliza casos para propor escalonamentos, mas adapta escalonamentos propostos por outros métodos (por exemplo, Sycara e Miyashita, 1994). Em outras palavras, casos podem englobar escalonamentos de fato (primeira abordagem) ou eles podem conter procedimentos de reparação (segunda abordagem).

Cunningham e Smyth (1996) examinam a primeira abordagem, explorando a reutilização de componentes de bons escalonamentos conhecidos em novos problemas de escalonamento. Isto envolve acumular uma base de casos com escalonamentos de boa qualidade, recuperar um ou mais casos similares para um novo problema de escalonamento e construir um novo escalonamento a partir de componentes dos casos recuperados. São descritas duas soluções RBC para o problema de escalonamento em uma máquina monoprocessadora. Estas soluções foram avaliadas, comparando-as com duas técnicas alternativas convencionais (*simulated annealing* e *myopic search*). Mostra-se que as duas abordagem utilizando RBC fornecem soluções de boa qualidade e melhoras significativas de tempo sobre a técnica *simulated annealing*.

Miyashita e Sycara (1995) implementaram o sistema CABINS, um esquema de aquisição de conhecimento e revisão iterativa para melhoramento de escalonamento e reparação reativa. Descreveram um método de aprendizagem baseado em casos para aquisição de preferências de otimização do usuário dependente do contexto e utilização destas preferências para melhorar incrementalmente a qualidade do escalonamento no gerenciamento de escalonamento preditivo e reativo em resposta a eventos de execução inesperados. Esta abordagem utiliza as preferências adquiridas do usuário para modificar dinamicamente o controle de busca para guiar o melhoramento do escalonamento.

## 2.3 Escalonamento de Processos em STRC

Conforme Melo Jr. (1993), existem diversas aplicações de computadores nas quais o processamento deve satisfazer restrições temporais rígidas, ou seja, deve ser garantido que

elas sejam completadas dentro de prazos especificados. Falhas em satisfazer estes prazos podem provocar uma degradação do sistema que, em algumas aplicações, pode ocasionar perda de vidas humanas ou grandes prejuízos financeiros. Sistemas nos quais a correção das saídas não depende apenas dos resultados lógicos dos processamentos, mas também dos instantes nos quais estes resultados são produzidos dentro dos prazos estabelecidos, são chamados de Sistemas de Tempo Real Crítico.

Sistemas de Tempo Real Crítico (STRC) têm uma variada gama de aplicações que vai de sistemas distribuídos complexos, responsáveis pelo tráfego aéreo de um país inteiro, até sistemas mais simples, como os presentes em motores de automóveis. Procura-se obter a correção lógica de STRC através de técnicas adequadas de especificação, projeto e verificação, enquanto a teoria de escalonamento busca atender ao cumprimento dos requisitos temporais especificados [Stankovic et al, 1985].

Em STRC com alguma complexidade, os processos a serem escalonados estão sujeitos a um grande número de restrições: tempo de pronto (instante mais cedo no qual o processo torna-se disponível para execução), tempo de execução, prazo de término (prazo máximo para finalização da execução de um dado processo), relações de precedência, e relações de exclusão mútua.

Um sistema de tempo real pode ser visto como consistindo de um sistema controlador e de um sistema controlado. Por exemplo, em uma fábrica automatizada, o sistema controlado é o chão de fábrica, com seus robôs, linhas de montagem etc, enquanto o sistema controlador é o computador e as interfaces homem-máquina que gerenciam e coordenam as atividades no chão de fábrica. O sistema controlado pode ser visto como o ambiente com o qual o computador interage.

O sistema controlador interage com o seu ambiente baseado nas informações que dispõe a seu respeito, provenientes, em geral, de sensores a ele conectados. É fundamental que o estado do ambiente, tal como percebido pelo sistema de controle, seja consistente com o seu estado efetivo, pois, caso contrário, a atuação do sistema de controle sobre o ambiente pode ser desastrosa. Portanto, faz-se necessário um monitoramento periódico do ambiente, bem como um processamento correto do ponto de vista lógico e temporal da informação sensoriada.

Devido às características do sistema controlado, o sistema controlador é constituído, geralmente, de diversas atividades distintas que necessitam ser executadas simultaneamente para poder acompanhar as mudanças de estado do sistema controlado. A realização de cada atividade do sistema controlador é feita através de um módulo de *software* específico chamado processo. Processos podem ser vistos como uma abstração de uma seqüência de código no qual, em qualquer ponto, podem ser caracterizados pelos processamentos e pelas informações de estado, ou seja, contador de programa, pilha e variáveis estáticas [Mok, 1983].

## *2.3.1 Aspectos que caracterizam STRC*

Em STRC, os diversos processos podem ser caracterizados sob vários aspectos, a saber: restrições de tempo, periodicidade, relações de precedência, relações de exclusão mútua, recursos, preempção e localização. A seguir, cada um destes aspectos é discutido brevemente [Xu e Parnas, 1990]. Sabe-se que o problema de escalonar um conjunto de processos com as restrições de tempo de pronto, de tempo de execução, prazo de término, relações de precedência e de exclusão mútua é *NP-Completo* [Xu e Parnas, 1991; Garey e Johnson, 1979].

***Restrições de Tempo***

As restrições de tempo de um processo podem ser especificadas através de vários parâmetros:

- ✧ Tempo de chegada ***a*** (*arrival time*): instante no qual o processo é invocado no sistema.

- ✧ Tempo de pronto ***r*** (*ready time*, *release time*): instante mais cedo no qual o processo torna-se disponível para execução. O tempo de pronto de um processo é maior que ou igual a seu tempo de chegada e, caso não existam restrições de recursos adicionais, tem-se $r = a$.

- ✧ Tempo de execução ***c*** (*computation time*): tempo máximo de execução de um processo. No cálculo deste parâmetro podem ser considerados alguns custos adicionais como, por exemplo, o custo de mudança de contexto.

- ✧ Prazo de término ***d*** (*deadline*): prazo máximo para o término de execução de um dado processo, representando a principal restrição de tempo. O tempo de término é um valor relativo ao tempo de chegada do processo. Dessa forma, o prazo de término absoluto ***da*** é dado por: $da=a+d$.

***Periodicidade***

Em relação à periodicidade, dependendo da natureza do tempo de chegada dos processos, estes podem ser classificados em: processos periódicos ou processos aperiódicos.

Processos periódicos são invocados uma vez a cada intervalo de tempo fixo, ou seja, exatamente uma vez por período $p$. Todas as restrições de tempo de processos periódicos são conhecidas previamente e, em particular, os tempos de chegada podem ser pré-determinados. Um exemplo típico são os processos que fazem varredura periódica de sensores. A Figura 2 ilustra as diversas instâncias de um processo periódico $P_i$, onde o prazo de término coincide com o período, ou seja, $d_i = p_i$.

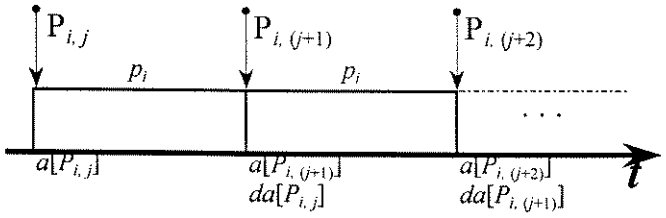



***Figura 2*** *Instâncias de um processo periódico*

Nota-se que o prazo de término da ($j$+1)-ésima instância do processo $P_i$, ou seja, o processo $P_{i,(j+1)}$ coincide com a invocação da instância ($j$+2), e o prazo de término de qualquer instância dá-se $P_i$ unidades de tempo após sua invocação. Dessa forma, tem-se que:

$$da[P_{i,j}] = a[P_{i,(j+1)}]$$
$$da[P_{i,j}] = a[P_{i,j}] + p_i$$

As relações matemáticas acima são válidas apenas em situações onde o prazo de término $d_i$ coincide com o período $p_i$. Em aplicações com restrições de tempo mais rígidas pode-se fazer necessário um prazo de término menor que o período. Obviamente, o prazo de término absoluto de qualquer processo $P_i$, independente da natureza de seu tempo de chegada, deve ser maior ou igual ao tempo de chegada do processo, acrescido de seu tempo de execução. Ou seja,

$$da[P_i] \geq a[P_i] + c[P_i].$$

Processos aperiódicos podem ser invocados a qualquer instante, não possuindo tempos de chegadas determinísticos. Estes processos estão geralmente associados a eventos aleatórios, tais como o aumento repentino de pressão e temperatura, e cujos instantes de ocorrência são imprevisíveis. Processos aperiódicos que possuem prazos para término de execução (*deadline*) são também chamados processos esporádicos.

### *Relações de Precedência*

Relações de precedência entre processos surgem quando um processo requer informações produzidas por outros processos. Neste caso, os processos deixam de ser independentes entre si, passando a existir uma ordem para a execução. Quando um processo **A** precede um processo **B**, somente após o término de **A** o processo **B** poderá iniciar sua execução.

### *Relações de Exclusão Mútua*

Quando um recurso qualquer do sistema (memória, arquivos, entrada e saída) é compartilhado entre vários processos, são necessárias relações de exclusão mútua entre eles para impedir o acesso simultâneo ao recurso compartilhado, o que poderia ocasionar inconsistências. Uma das maneiras de se assegurar exclusão mútua é o uso de primitivas do tipo semáforo [Ben-Ari, 1990].

***Recursos***

Dependendo da atividade realizada por um processo, este pode necessitar de recursos específicos para poder executar, tais como unidades de entrada e saída e processadores dedicados. Neste caso, o tempo de pronto do processo é geralmente posterior ao tempo de chegada do mesmo, pois, normalmente, leva-se algum tempo para alocar todos os recursos necessários. Em muitos STRC, os recursos requeridos por um processo são assumidos estar disponíveis assim que o processo é invocado. Para estes sistemas, os processadores são os recursos primários.

***Preempção***

Em relação à preempção em STRC, os processos podem ser preemptíveis ou não-preemptíveis, dependendo da natureza da aplicação. Um processo é dito preemptível se sua execução pode ser interrompida por outros processos de maior prioridade em qualquer instante, retornando sua execução mais tarde no ponto em que havia parado. Processos não preemptíveis, uma vez iniciados, executam até o seu término ou até liberarem explicitamente o processador.

***Localização***

Quanto à localização, em um sistema onde existem múltiplos processadores, os processos devem ser alocados a estes, considerando diversos aspectos, como o balanceamento de carga de processamento, minimização dos custos de comunicação, e requisitos de tolerância a falhas e de recursos adicionais.

## *2.3.2 Algoritmos de Escalonamento*

O sistema controlador de um sistema de tempo real é constituído de diversos processos que necessitam ser executados simultaneamente para acompanhar a dinâmica do sistema controlado. Normalmente, o número de processos do sistema supera o número de processadores. Sempre que isto ocorre, surge a necessidade de um componente de *software*, o escalonador, que implementa um ou mais algoritmos de escalonamento para coordenar a execução dos processos.

Um algoritmo de escalonamento é um conjunto de regras que determina o processo a ser executado em um determinado momento. Processos são entidades escalonáveis em STRC. A função de um algoritmo de escalonamento é determinar, para um dado conjunto de processos, se uma ordem (a seqüência e os tempos) para execução dos processos existe, tal que as restrições de tempo, precedência, exclusão mútua, recursos e outras, que porventura existam, sejam satisfeitas, e calcular tal ordem se ela existir [Cheng e Stankovic, 1987]. Em muitas situações, o problema de encontrar um escalonamento para um conjunto de processos em STRC é bastante árduo devido à necessidade de se satisfazer os requisitos de tempo

individuais dos processos (principalmente o prazo de término), os quais são ditados pelos fenômenos físicos controlados. Portanto, em STRC, o escalonamento de processos é o problema mais importante, porque é o algoritmo de escalonamento que garante que os processos cumprirão seus prazos de término. Uma taxonomia dos algoritmos de escalonamento para sistemas de tempo real (STR) é apresentada na Figura 3 [Cheng e Stankovic, 1987].

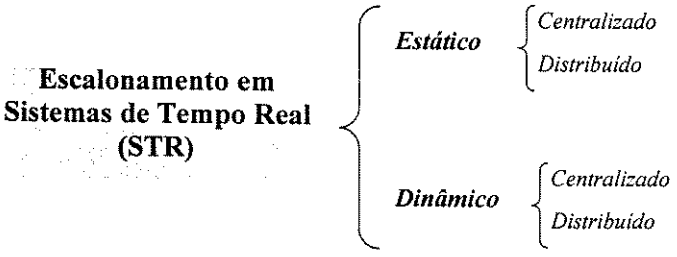



***Figura 3*** *Uma taxonomia de algoritmos de escalonamento para STR*

Algoritmos de escalonamento para STRC podem ser estáticos ou dinâmicos. A abordagem estática calcula o escalonamento dos processos *off-line*, e requer um conhecimento prévio completo das características dos processos. A abordagem dinâmica escalona os processos *on-line*, e permite que processos possam ser invocados dinamicamente. Abordagens estáticas, embora apresentem um baixo custo em tempo de execução, são inflexíveis e não se adaptam a ambientes cujo comportamento não são completamente previsíveis. As abordagens dinâmicas são mais flexíveis e podem se adaptar às mudanças do ambiente, mas apresentam um custo em tempo de execução maior, e podem não ser capazes de assegurar que restrições de tempo serão cumpridas.

De acordo com a arquitetura do sistema alvo, os algoritmos de escalonamento podem ser classificados em centralizados ou distribuídos. Um sistema centralizado é aquele no qual os processadores estão localizados em um único ponto do sistema, e o custo de comunicação entre os processadores é desprezível, comparado ao custo de execução. Sistemas multiprocessadores com memória compartilhada e sistemas monoprocessadores são exemplos de sistemas centralizados. Já nos sistemas distribuídos, os processadores estão localizados em diferentes pontos do sistema, e o custo de comunicação entre processadores é um importante fator que deve ser explicitamente levado em conta no escalonamento. Nesta última classe se enquadram, por exemplo, as redes locais de computadores [Melo Jr., 1993].

A escolha do processo a ser executado é feita pelo escalonador, que consiste de um módulo do núcleo de tempo real ou do sistema operacional. Um escalonador deve coordenar o uso de todos os recursos do sistema usando um conjunto de algoritmos de escalonamento (no mínimo um), que satisfaçam os seguintes objetivos [Sprunt et al, 1989; Sha e Goodenough, 1990]:

- ✧ Garantir que os processos cumpram seus prazos de término.

- ✧ Fornecer bons tempos médios de resposta para processos aperiódicos que não possuem prazo de término.

- ✧ Garantir estabilidade em momentos de sobrecarga transitória do sistema. Quando o sistema está sobrecarregado por diversos eventos e é impossível satisfazer todos os prazos de término, deve-se, ainda, garantir os prazos de término de processos críticos previamente selecionados. Um escalonador com esta característica é dito ser estável.

- ✧ Obter um alto grau de escalonabilidade ($G$). Dado um conjunto de $n$ processos periódicos $P_1, P_2, \ldots, P_n$, onde cada processo $P_i$ possui um tempo de execução $c_i$ e um período $p_i$, o fator de utilização de processador ($U$) pelos $n$ processos é definido como sendo:

$$U(n) = \sum_{i=1}^{n} \frac{c_i}{p_i}$$

A qualidade de escalonadores para STRC é julgada de acordo com os objetivos citados. Para um dado conjunto de processos, um escalonamento é dito ser factível se os processos são escalonados de modo que todos cumpram seus prazos de término.

## 2.4 Solução do Problema de Escalonamento Utilizando o EPPE de Melo Jr. (1993)

Considerando-se um conjunto de processos sujeitos a restrições de tempo de pronto, tempo de execução, prazo de término e relações arbitrárias de precedência e exclusão mútua para Sistemas de Tempo Real Crítico (STRC), Xu e Parnas (1990) apresentaram um algoritmo para encontrar, todas as vezes que existir, um escalonamento no caso de um conjunto de processos preemptivos em um ambiente monoprocessador, tal que cada processo inicie sua execução após seu tempo de pronto e termine no máximo até o seu prazo de término, satisfazendo todas as relações de precedência e exclusão mútua, definidas sobre pares ordenados de segmentos de processos.

O algoritmo tem como objetivo encontrar um escalonamento sempre que uma solução existir. Este problema possui complexidade exponencial. O algoritmo faz uso de uma técnica de enumeração implícita *branch and bound*, que na maioria dos casos apresenta um desempenho muito melhor que uma enumeração completa, mas possui complexidade exponencial [French, 1986]. Com o uso do método *branch and bound*, apesar de se considerar todas as possibilidades de escalonamento, isto não é feito de modo explícito, pois o método explora a árvore de busca, inteligentemente, fazendo uso da idéia de *bounding* [French, 1986].

A utilização de um algoritmo de escalonamento *off-line* para STRC pode reduzir significativamente os custos de escalonamento e mudanças de contexto em tempo de execução. Mais importante que isto, todas as relações de precedência e exclusão mútua já estão asseguradas no escalonamento obtido *off-line*, não sendo necessário o uso de primitivas de sincronização em tempo de execução, e tem-se, ainda, a garantia prévia de que os prazos de término serão satisfeitos. Isto aumenta a previsibilidade do sistema e diminui a possibilidade de *deadlocks* [Melo Jr., 1993]. A seguir, são introduzidas algumas definições e notações usadas pelo algoritmo *off-line* de Xu e Parnas (1990), que facilitam o entendimento do algoritmo discutido.

Intuitivamente, uma unidade de segmento é a menor porção indivisível de um processo. O conjunto de processos a ser escalonado será denotado por **P**. Cada processo $\mathbf{p} \in \mathbf{P}$ consiste de uma seqüência finita de segmentos p[0], p[1], ..., p[$n$], onde p[0] é o primeiro segmento e p[$n$] é o último segmento do processo **p**.

Para cada segmento ***i*** define-se:

- ✧ um tempo de pronto ***r*[*i*]**;
- ✧ um *deadline* (prazo de término) absoluto ***da*[*i*]**; e
- ✧ um tempo de execução ***c*[*i*]**.

Assume-se que os parâmetros $r[i]$, $da[i]$, $c[i]$ possuem valores inteiros não negativos, dados em múltiplos de uma unidade de tempo básica do sistema.

O conjunto de todos os segmentos pertencentes a processos em **P** será denotado por **S(P)**.

O algoritmo de Xu e Parnas (1990) é bastante flexível, pois permite modelar situações reais, onde algumas porções de um processo são preemptíveis por certas porções de outros processos, enquanto outras não são em relação a outros processos. Essas porções de processos correspondem aos segmentos. Inicialmente, todos os processos de **P** devem ser decompostos em segmentos para se obter o conjunto **S(P)**, que constitui a entrada do algoritmo, juntamente com as relações de precedência e exclusão mútua definidas entre pares ordenados de segmentos de **S(P)**.

A partir do tempo de pronto, tempo de execução e *deadline* (prazo de término) absoluto de cada processo e do tempo de execução e tempo de início de cada segmento relativo ao início do processo contendo o segmento, pode-se calcular o tempo de pronto, tempo de execução e prazo de término absoluto para cada segmento.

Como exemplo, considera-se um processo $\mathbf{A} \in \mathbf{P}$ com tempo de pronto $r = 2$, tempo de execução $c = 7$ e *deadline* absoluto $da = 12$. O processo **A** acessa um recurso $R_1$ qualquer, compartilhado com outros processos do sistema. A decomposição do processo em segmentos é apresentada na Figura 4. Inicialmente, faz-se o prazo de término do último segmento como sendo o prazo de término do processo e o tempo de pronto do primeiro segmento igual ao tempo de pronto do processo. Os demais valores de parâmetros são obtidos de modo direto.

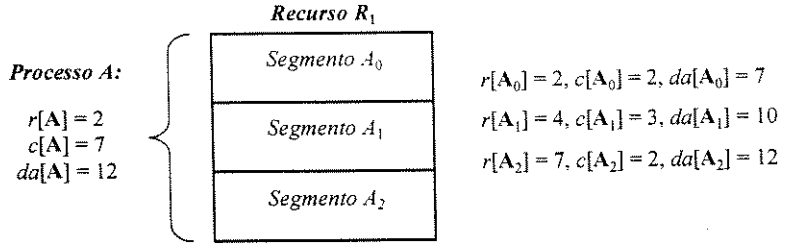



***Figura 4*** *Decomposição de um processo em segmentos*

Cada unidade de segmento requer uma unidade de tempo para executar, durante a qual ele não pode sofrer preempção por qualquer outro processo. As relações de precedência e exclusão mútua das unidades de segmentos possuem as relações, que devem ser satisfeitas no problema original. Um escalonamento praticável de um conjunto de processos **P** é um escalonamento válido de **P**, tal que o valor de seu retardo seja menor ou igual a zero, ou seja, todos os segmentos de **S(P)** cumprem seus *deadlines* (prazos de término). O escalonamento de processos periódicos apresenta uma propriedade fundamental: o caráter cíclico [Leung e Merril, 1980]. Esta propriedade permite estabelecer um limite de tempo a partir do qual a seqüência de execução dos processos periódicos se repete. O valor deste limite de tempo, aqui denominado tamanho da janela cíclica, será denotado por *w*, que é o mínimo múltiplo comum entre os períodos dos processos envolvidos em um problema de escalonamento.

Para se encontrar uma solução praticável ou ótima usa-se uma técnica de enumeração implícita *branch and bound*, discutida a seguir [Melo Jr., 1993].

O problema de escalonar um conjunto de processos sujeitos a restrições de tempo de pronto, tempo de execução, prazo de término e relações de precedência e exclusão mútua arbitrárias é conhecido ser *NP-Completo*, o que, efetivamente, exclui a possibilidade de existência de algum algoritmo de complexidade polinomial para solucionar o problema [Xu e Parnas, 1990; Garey e Johnson, 1979]. Para se alcançar o objetivo de encontrar um escalonamento ótimo ou praticável sempre que algum existir, faz-se necessário usar um método enumerativo, o qual, simplesmente, após listar ou enumerar todas as possíveis soluções, elimina os escalonamentos não ótimos ou não válidos da lista.

Um método enumerativo pode ser do tipo explícito ou implícito. Na enumeração explícita, procede-se uma enumeração completa de todas as possíveis soluções e, em seguida, escolhe-se a melhor segundo algum critério. As técnicas de enumeração implícita, apesar de também considerarem todas as soluções, exploram de modo inteligente o conjunto de soluções na tentativa de reduzir o esforço de busca.

Para buscar uma solução válida praticável ou ótima, o algoritmo de Xu e Parnas (1990) usa uma técnica de enumeração implícita *branch and bound*. No processo de busca, o método *branch and bound* gera uma árvore de busca que tem em seu nó raiz uma solução válida inicial. Cada nó da árvore de busca, tal como a raiz, corresponde a uma solução válida

completa, mas não necessariamente praticável. Cada nó filho representa uma maneira diferente de reduzir o retardo do segmento mais retardado da solução válida de seu nó pai. Para reduzir o esforço de busca, um procedimento de *bounding* é usado para descartar nós filhos que não podem levar a uma solução melhor do que a melhor solução encontrada até o momento na busca. O procedimento de *bounding* é realizado calculando um *lowerbound* do retardo do escalonamento para cada nó filho. O valor de *lowerbound* de um nó é uma estimativa para o menor valor de retardo de escalonamento possível que pode ser obtido em alguma solução válida de seus nós descendentes. No processo de busca, comparando-se o valor do retardo da melhor solução válida obtida até o momento com o valor do *lowerbound* de um nó, determina-se a necessidade de continuar ou não explorando o ramo da árvore. Virtualmente, métodos *branch and bound* podem ter que explorar completamente todos os nós da árvore de busca, sendo tão ruins quanto uma enumeração completa, mas, em geral, apresentam um desempenho bem melhor [French, 1986].

A eficiência dos métodos *branch and bound* tem uma relação direta com a qualidade do *lowerbound*. Um *lowerbound* é considerado bom quando ele não é muito menor do que o menor valor real da medida de desempenho, eliminando uma grande quantidade de nós em níveis de profundidade da árvore de busca mais altos, reduzindo substancialmente o esforço de busca. Contudo, deve haver um compromisso entre a qualidade do *lowerbound* e o esforço para sua computação.

Melo Jr. (1993), para validar sua estratégia de escalonamento de processos periódicos e esporádicos em Sistemas de Tempo Real Crítico monoprocessados, implementou o sistema Escalonador de Processos Periódicos e Esporádicos (EPPE), que apresenta o serviço de escalonamento *off-line*. Este serviço está baseado no algoritmo de Xu e Parnas (1990) descrito anteriormente.

### *2.4.1 Um Exemplo de Problema de Escalonamento*

Considere os processos periódicos **A**, **B**, **C** e **D** definidos na Tabela 1 e representados na Figura 5. Para uma instância do processo **A**, por exemplo, o prazo de término absoluto ***da*** ocorrerá 30 unidades de tempo após sua invocação. O processo **B** possui dois pontos de sincronismo com o processo **A**, o que faz com que o processo **B** seja decomposto em dois segmentos: **B0** e **B1**. O segmento **B0** e o processo **D** acessam um mesmo recurso $R_1$ qualquer. Considere que as primeiras instâncias de todos os processos periódicos possuem tempo de chegada igual a zero. $\mathbf{B0}_1$ é o segmento **B0** da primeira instância do processo **B**, **PC** indica uma relação de precedência e **EX** uma relação de exclusão mútua. Note que as relações não são simétricas.

O tamanho da janela cíclica $w$ para este conjunto de processos periódicos é dado pelo mínimo múltiplo comum entre os períodos dos processos apresentados na Tabela 1, como segue:

$$w = \text{m.m.c.}\{30, 60, 40, 60\} \Rightarrow w = 120$$

Logo, para se obter o conjunto **S(P)**, deve-se decompor em segmentos todas as instâncias de processos periódicos que ocorrem no intervalo $[0, w] = [0, 120]$. As relações de precedência e exclusão mútua devem ser definidas entre pares ordenados de segmentos **S(P)**. Estas relações, juntamente com o conjunto **S(P)**, constituem a entrada do algoritmo de Xu e Parnas (1990), conforme descrito na seção anterior.

***Tabela 1*** *Restrições de tempo para o exemplo de escalonamento*

| **Processos Periódicos** | ***c*** | ***p*** | ***da*** |
|---|---|---|---|
| A | 6 | 30 | 30 |
| B | 8 | 60 | 60 |
| C | 8 | 40 | 40 |
| D | 12 | 60 | 60 |

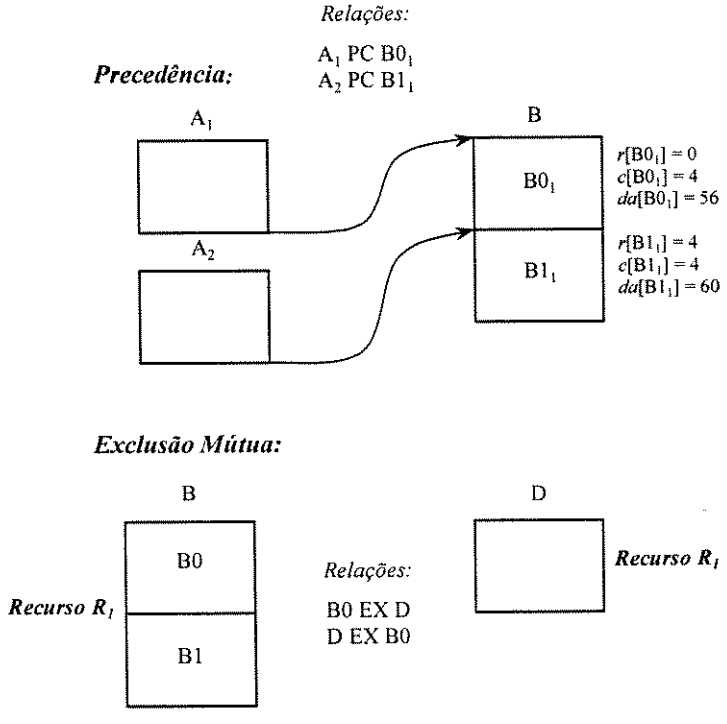



***Figura 5*** *Processos periódicos para o exemplo de escalonamento*

Após a decomposição dos processos em segmentos, o problema pode então ser descrito através dos dados da Tabela 2.

***Tabela 2*** *Decomposição em segmentos para o exemplo de escalonamento*

| Processo | Segmento | Identificador | *r* | *c* | *da* |
|---|---|---|---|---|---|
| **A** | $A_1$ | 1 | 0 | 6 | 30 |
| | $A_2$ | 2 | 30 | 6 | 60 |
| | $A_3$ | 3 | 60 | 6 | 90 |
| | $A_4$ | 4 | 90 | 6 | 120 |
| **B** | $B0_1$ | 5 | 0 | 4 | 56 |
| | $B1_1$ | 6 | 4 | 4 | 60 |
| | $B0_2$ | 7 | 60 | 4 | 116 |
| | $B1_2$ | 8 | 64 | 4 | 120 |
| **C** | $C_1$ | 9 | 0 | 8 | 40 |
| | $C_2$ | 10 | 40 | 8 | 80 |
| | $C_3$ | 11 | 80 | 8 | 120 |
| **D** | $D_1$ | 12 | 0 | 12 | 60 |
| | $D_2$ | 13 | 60 | 12 | 120 |

Além disso, são consideradas as seguintes relações de precedência e exclusão mútua:

- ✧ *Precedência*: 1 PC 5, 2 PC 6, 3 PC 7, 4 PC 8
- ✧ *Exclusão mútua*: 5 EX 12, 7 EX 13, 12 EX 5, 13 EX 7

## *2.4.2 Solução do Exemplo Utilizando o EPPE de Melo Jr. (1993)*

Após o processamento dos dados de entrada apresentados na Figura 5 e na Tabela 2, gera-se o escalonamento praticável para os processos periódicos, o qual é apresentado na Figura 6, ao longo da primeira janela cíclica, por meio de um diagrama de Gantt [Melo Jr., 1993].

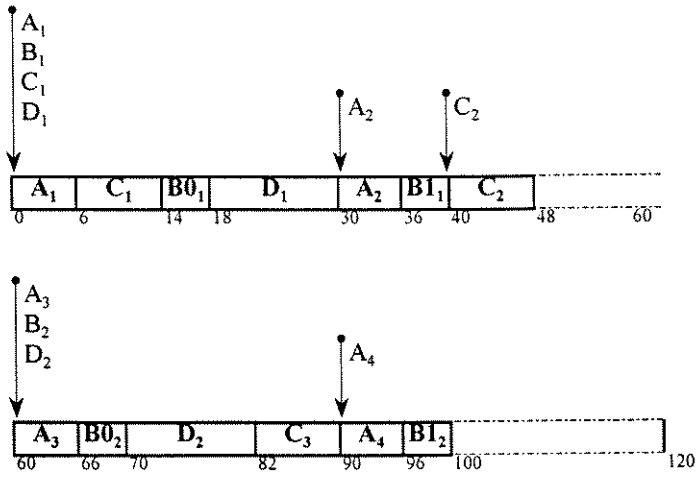



***Figura 6*** *Escalonamento gerado pelo EPPE de Melo Jr. (1993) para os processos do exemplo*

De acordo com o gráfico de Gantt pode-se determinar os períodos em que cada tarefa é tratada pelo monoprocessador. Pode-se observar que todas as relações de precedência e exclusão mútua estão satisfeitas no escalonamento obtido, não se fazendo necessários mecanismos de sincronização em tempo de execução.

## 2.5 Descrição do Problema de Escalonamento Abordado

O problema a ser resolvido é o escalonamento de processos periódicos em um STRC monoprocessado, cujo contexto motiva uma solução baseada em casos. Tipicamente, escalonamentos devem ser produzidos com determinada freqüência e o conjunto de processos a ser escalonado pode ser similar a um outro conjunto escalonado anteriormente [Cunningham e Smyth, 1996]. Em particular, o problema considerado neste trabalho possui as seguintes características: restrições de tempo, periodicidade e relações de precedência. No capítulo anterior, tais características de um STRC foram discutidas de forma genérica. A seguir, cada um destes aspectos considerados para o problema proposto é colocado brevemente.

As restrições de tempo consideradas para o escalonamento de processos periódicos em um STRC monoprocessado são as seguintes:

- Tempo de pronto ***r*** (*ready time*, *release time*): instante mais cedo no qual o processo torna-se disponível para execução.
- Tempo de execução ***c*** (*computation time*): tempo máximo de execução de um processo.
- Prazo de término ***d*** (*deadline*): prazo máximo para término de execução de um dado processo.

Os processos a serem escalonados considerados neste trabalho são processos periódicos.

## 2.6 O Problema de Escalonamento em STRC e o RBC

Um problema constituído por subproblemas que podem ser resolvidos separadamente levando à solução do todo é uma premissa importante a ser considerada, pois pode-se utilizar problemas resolvidos similares a estes subproblemas nas suas resoluções. Um algoritmo dedicado, implementado através do EPPE [Melo Jr., 1993], propõe-se a resolver um problema de escalonamento em STRC. Estes problemas podem ser complexos de maneira que o tempo para resolvê-lo torna-se grande. Tentar diminuir este tempo é um objetivo. Ter armazenado em uma base problemas resolvidos anteriormente similares a subproblemas de problemas novos possibilita o emprego do escalonamento baseado em casos. O esforço necessário para o

EPPE resolver um problema complexo pode ser reduzido se houver problemas anteriores que possibilitem a sua simplificação previamente. Ainda pode-se encontrar uma solução direta para o problema a partir dos problemas armazenados na base.

## 2.7 Conclusões

Neste capítulo, a metodologia Raciocínio Baseado em Casos (RBC) foi introduzida, destacando-se que RBC é um modelo de raciocínio, que envolve o entendimento de situações, a resolução de problemas e a aprendizagem, integrando esses aspectos a processos de memória [Kolodner, 1993]. Ainda, segundo Kolodner (1993), um caso é uma peça de conhecimento contextualizado, representando uma experiência que ensina uma lição fundamental para alcançar os objetivos do módulo de inferência (raciocínio). O processo de RBC envolve os passos de recuperação de casos, proposta de um esboço de solução, adaptação, raciocínio avaliativo (justificativa e crítica), teste avaliativo e atualização da memória de casos. Foram apresentadas as vantagens e desvantagens em se usar RBC como metodologia para a construção de sistemas de resolução de problemas. Alguns sistemas descritos na literatura foram apresentados, ilustrando o potencial do RBC.

Foram discutidas abordagens baseadas em casos para o problema de escalonamento e descrito os aspectos do problema de escalonamento de processos em Sistemas de Tempo Real Crítico (STRC) e introduzido o Escalonador de Processos Periódicos e Esporádicos (EPPE) de Melo Jr. (1993), que se baseia no algoritmo de Xu e Parnas (1990), para a solução deste problema. As características consideradas para a abordagem considerada foram descritas e destacou-se a relevância do uso do RBC para a resolução do problema de escalonamento.

# 3

# Escalonamento Baseado em Casos

O objetivo deste capítulo é apresentar a arquitetura de um sistema baseado no método Raciocínio Baseado em Casos (RBC) para a resolução de problemas de escalonamento de processos em um Sistema de Tempo Real Crítico (STRC) monoprocessado e discutir os principais aspectos de sua implementação. A seção 2.5 do capítulo 2 apresenta as características consideradas neste trabalho para o problema de escalonamento abordado. Foi desenvolvido um sistema, que é a principal contribuição deste trabalho, o qual se baseia em casos resolvidos no passado para resolver novos problemas e é capaz de aprender novos casos. O Sistema de Escalonamento Baseado em Casos (SEBC), implementado neste trabalho, possui como característica principal a idéia de simplificação de um problema através de substituições de seus subproblemas similares a problemas armazenados em uma base de casos e adaptação das soluções destes problemas na busca de uma solução para o problema apresentado.

Dois módulos importantes para o SEBC e que são implementados por Melo Jr. (1993) e Messmer (1995) estão sendo utilizados neste trabalho. O Escalonador de Processos Periódicos e Esporádicos (EPPE) de Melo Jr. (1993) representa um algoritmo dedicado na resolução de um problema de escalonamento de processo em STRC monoprocessado e o GUB (implementação para casamento de grafo) de Messmer (1995) é responsável pela recuperação de casos similares na base de casos. O EPPE é descrito no capítulo 2 e o GUB é descrito em linhas gerais na seção 3.2 deste capítulo.

Embora os problemas de escalonamento em STRC possam ser resolvidos através de um algoritmo dedicado, o tempo necessário para resolvê-lo aumenta à medida que os problemas crescem. Tentar diminuir este tempo é um objetivo que o SEBC se propõe a alcançar.

As seções 3.1 e 3.2 introduzem as características de representação do problema de escalonamento e de recuperação de problemas similares na base de casos, consideradas pelo SEBC. A seção 3.3 descreve e discute os aspectos de implementação do SEBC e apresenta seu ciclo para resolução de problemas baseado em casos. Os itens 3.3.1 e 3.3.2 descrevem os processos de busca de casos na base e simplificação do problema novo e aprendizagem de problemas e soluções.

# 3.1 Representação do Problema de Escalonamento

Geralmente, grafos podem ser utilizados para descrever objetos, cujas partes são representadas através de vértices e as relações existentes são representadas através de arcos do grafo [Messmer, 1995]. Rótulos simbólicos e atributos numéricos são atribuídos aos vértices e arcos para descrever as propriedades das partes e suas relações. Um grafo dirigido com atributos é uma sêxtupla

$$G = (V, E, L, A, \mu, \nu)$$

na qual

- $V = \{v_1, v_2, \ldots, v_n\}$ é o conjunto de vértices;
- $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ é o conjunto de arcos;
- $L$ é um conjunto finito de rótulos simbólicos;
- $A$ é um conjunto finito de valores de atributos;
- $\mu: V \rightarrow L \times A_k$ é a função de rotulação do vértice e
- $\nu : E \rightarrow L \times A_k$ é a função de rotulação do arco.

O problema de escalonamento é representado neste trabalho através de grafos acíclicos dirigidos com atributos. Os vértices do grafo representam os processos, seus rótulos indicam quais seus tempos de execução (característica particular a este trabalho, descrita na seção 3.2) e seus atributos trazem os tempos de pronto e prazo de término. Os arcos determinam as relações de precedência.

Na Figura 7, tem-se a representação de um problema de escalonamento através de um grafo, com as características descritas na Tabela 3.

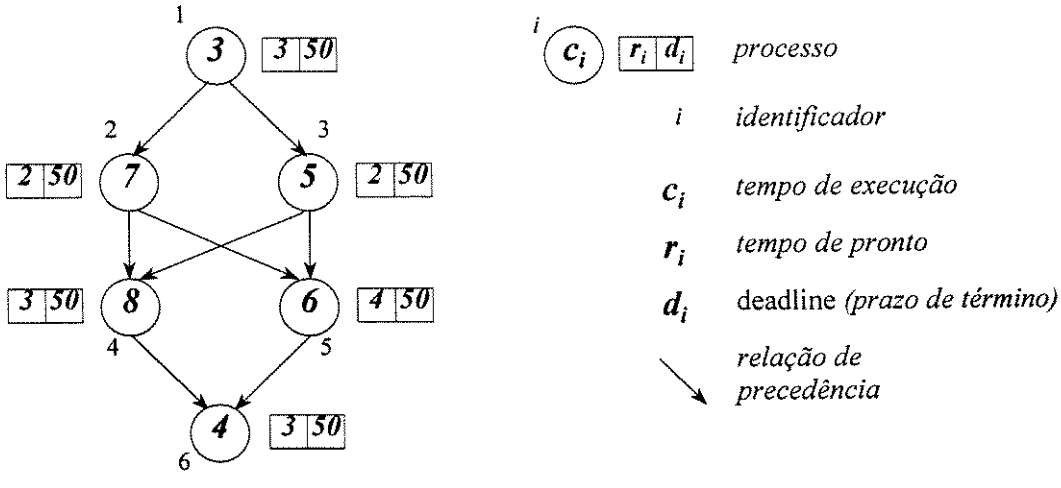



***Figura 7*** *Representação de um problema de escalonamento através de grafo*

***Tabela 3*** *Restrições de tempo e relações entre as tarefas para o problema de escalonamento representado através de grafo*

| Processo | Identificador | *r* | *c* | *d* | Precedência |
|---|---|---|---|---|---|
| A | 1 | 3 | 3 | 50 | ② ③ |
| B | 2 | 2 | 7 | 50 | ④ ⑤ |
| C | 3 | 2 | 5 | 50 | ④ ⑤ |
| D | 4 | 3 | 8 | 50 | ⑥ |
| E | 5 | 4 | 6 | 50 | ⑥ |
| F | 6 | 3 | 4 | 50 | — |

A Figura 7 ilustra um exemplo de grafo de tarefas. Sendo $V$ um conjunto de tarefas (representadas pelos vértices), a relação $v_i < v_j$ implica que a computação da tarefa $v_j$ depende dos resultados da computação da tarefa $v_i$. Em outras palavras, a tarefa $v_i$ deve ser computada antes da tarefa $v_j$ e o resultado da computação de $v_i$ deve ser conhecido pelo processador na computação de $v_j$. Estão associados às tarefas o tempo de execução, além do seu tempo de pronto e prazo de término. Cada arco representa a relação de precedência entre duas tarefas. Assim, tem-se uma representação estruturada do problema. Este tipo de representação de relações complexas é mais natural (intuitiva) para as pessoas.

Um subproblema é uma parte de um problema maior que pode ser resolvida separadamente sem prejuízo para a solução do problema como um todo. Os problemas armazenados na base de casos e os problemas apresentados ao SEBC são representados estruturalmente por meio de grafos acíclicos dirigidos com atributos.

Tem-se a formação de um grupo quando um subproblema é composto por vértices que possuam como precededentes direta ou indiretamente um mesmo vértice $v_e$ (chamado aqui de vértice de entrada) e tenham como sucessores direta ou indiretamente um mesmo vértice $v_s$ (chamado aqui de vértice de saída). É necessário formar um grupo para que o subproblema possa ser substituído por um único novo processo, que terá como precedente o processo precedente ao vértice de entrada e precederá o processo que o vértice de saída precede, não descaracterizando o problema inicial, isto é, as relações de precedência. A Figura 8 ilustra esta situação.

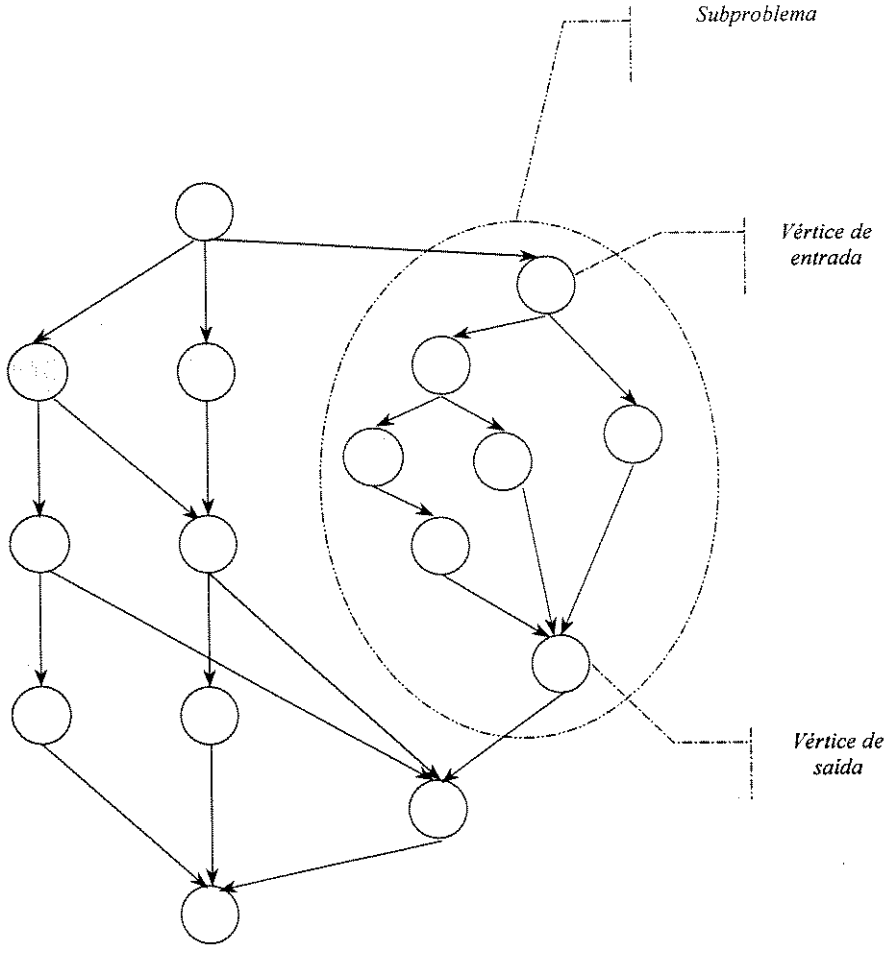



***Figura 8*** *Ilustração de um subproblema que forma um grupo em um grafo representando um problema novo*

## 3.2 Recuperação de Problemas Similares

Para se utilizar um problema resolvido anteriormente, armazenado na base de casos, é necessário que se verifique se ele é aplicável à nova situação apresentada. Esta verificação é feita através do casamento (*matching*) ou da avaliação de similaridade (*similarity-assessment*) entre os problemas na base e os subproblemas do problema apresentado ao SEBC. Como estão sendo utilizados grafos para a representação dos problemas, é avaliada a similaridade de grafos, ou seja, a similaridade estrutural.

Quando grafos são utilizados para representar problemas armazenados em uma base, os passos necessários para recuperá-los incluem algum tipo de casamento de grafos ou a identificação de grupos isomórficos [Myaeng e Lopez-Lopez, 1992; Maurer, 1993].

Formalmente, $G$ é um isomorfismo de subgrafo de $G'$ se todos os nós e arcos de $G$ estão contidos em $G'$, e se os nós e arcos correspondentes possuem o mesmo rótulo. A Figura 9 ilustra um casamento de grafo (verifica-se que o grafo $G$ é um subgrafo do grafo $G'$) para recuperação de problemas similares estruturalmente na base de casos. A detecção de isomorfismo de subgrafos é de relevância prática para situações em que os problemas podem ser decompostos em subproblemas que podem ser resolvidos individualmente [Messmer, 1995]. Nestas situações, pode-se armazenar um conjunto de problemas resolvidos previamente em uma base de casos passados; esta base é representada por casos $cp_1$, ..., $cp_n$ (problema-solução $j$). A partir desta base, dado um novo problema $cn$, deve-se buscar um casamento entre ele e cada caso $cp_j$ armazenado na base. Se $cn$ possui um ou mais $cp_j$ como subgrafos, conclui-se que as soluções destes subproblemas podem ser utilizadas para o novo problema $cn$.

Como o processo de recuperação de casos na base exige um algoritmo de verificação da similaridade entre o problema novo e os problemas armazenados, o SEBC utiliza o algoritmo de Ullman (1976). O algoritmo de detecção de isomorfismo de subgrafo proposto por Ullman (1976) e implementado por Messmer (1995) através da GUB (*Graph University of Bern*, uma implementação para casamento de grafo) fornece o mecanismo de recuperação de problemas similares estruturalmente aos subproblemas do problema novo.

Basicamente, dado um grafo $G$ e um grafo $G'$, o algoritmo de detecção de isomorfismo de subgrafo de Ullman (1976) tenta encontrar todos os isomorfismos de grafo ou subgrafo de $G$ em $G'$. Utiliza-se um procedimento de enumeração simples, no qual todos os mapeamentos possíveis a partir de vértices de $G$ para vértices de $G'$ são gerados e subseqüentemente testados para as condições de isomorfismo de grafo ou subgrafo. Porém, o número de mapeamentos possíveis é altamente combinatorial e, portanto, tal procedimento não é aplicável na prática. A fim de reduzir o número de mapeamentos que devem ser testados, é melhor iniciar com um vértice único para mapeamento de vértice e então gradualmente estender este mapeamento tal que a função do casamento resultante sempre denote um isomorfismo de subgrafo. Se, em algum ponto, o mapeamento não representa um isomorfismo de subgrafo, então o processo retorna (*backtracks*). Isto é, um vértice modelo mapeado previamente é reatribuído a outro vértice entrada e as condições para isomorfismo de subgrafo são testadas novamente. O GUB foi alterado para sua utilização pelo SEBC quanto ao aspecto de verificação do rótulo dos vértices. A função original verifica se o valor numérico de um rótulo de $G$ é igual ao de $G'$. A alteração efetuada faz com que seja verificado se o valor numérico do rótulo de $G'$ é menor ou igual ao de $G$. Esta modificação possibilita o refinamento do processo de recuperação, pois, após o passo de verificação de similaridade estrutural, os valores de tempo de execução, tempo de pronto e prazo de término teriam que ser analisados para validar a utilização dos casos recuperados.

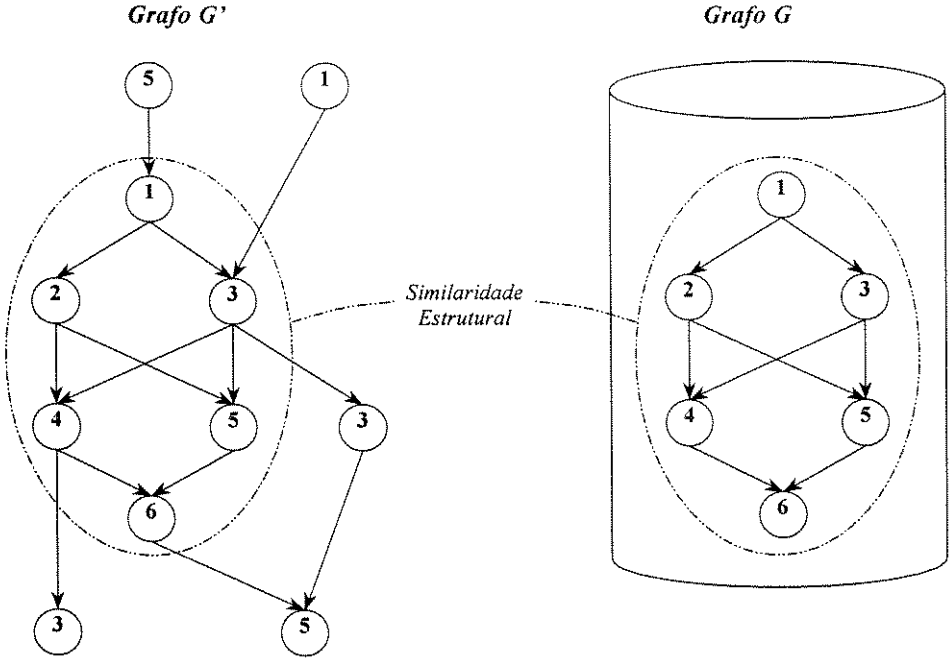



***Figura 9*** *Exemplo de similaridade estrutural*

# 3.3 Aspectos de Implementação

A Figura 10 apresenta a arquitetura do Sistema de Escalonamento Baseado em Casos (SEBC) proposto neste trabalho, cujo funcionamento é discutido resumidamente a seguir.

Um problema novo de escalonamento é apresentado ao SEBC, que verifica a existência na base de casos de problemas similares estruturalmente aos subproblemas deste novo problema.

Se não existir pelo menos um problema similar na base, o processo de aprendizagem é chamado e o problema simplificado até o momento é passado como parâmetro.

Se houver um ou mais problemas com estruturas similares, verifica-se se as condições de formação de grupo para o subproblema e as relações de tempo de pronto e prazo de término da Tabela 4 para os problemas recuperados são atendidas. Se estas condições não forem satisfeitas, não é possível proceder à substituição do subproblema por um único processo e continuar o processo de uma nova recuperação até que o problema simplificado

apresente uma única tarefa. Nesta situação, o processo de aprendizagem também é chamado e o problema simplificado até o momento é passado como parâmetro.

Quando houver a recuperação de problemas similares estruturalmente e soluções reutilizáveis e subproblemas substituíveis, o SEBC procede à substituição do subproblema por um novo processo que utiliza a solução adaptada do problema similar recuperado da base.

Se o problema simplificado possuir apenas uma tarefa, significa que o problema apresentado foi resolvido de forma direta com o conhecimento disponível na base de casos e a solução é apresentada. Se ainda não foi possível resolvê-lo, o problema simplificado é apresentado novamente para o processo de recuperação, verificação de atendimento a condições e substituição até que a solução seja encontrada através da base de casos ou o fluxo seja desviado para o processo de aprendizagem.

O processo de aprendizagem do SEBC resolve através do algoritmo dedicado o problema simplificado até o momento e armazena este problema e a solução encontrada na base de casos do sistema.

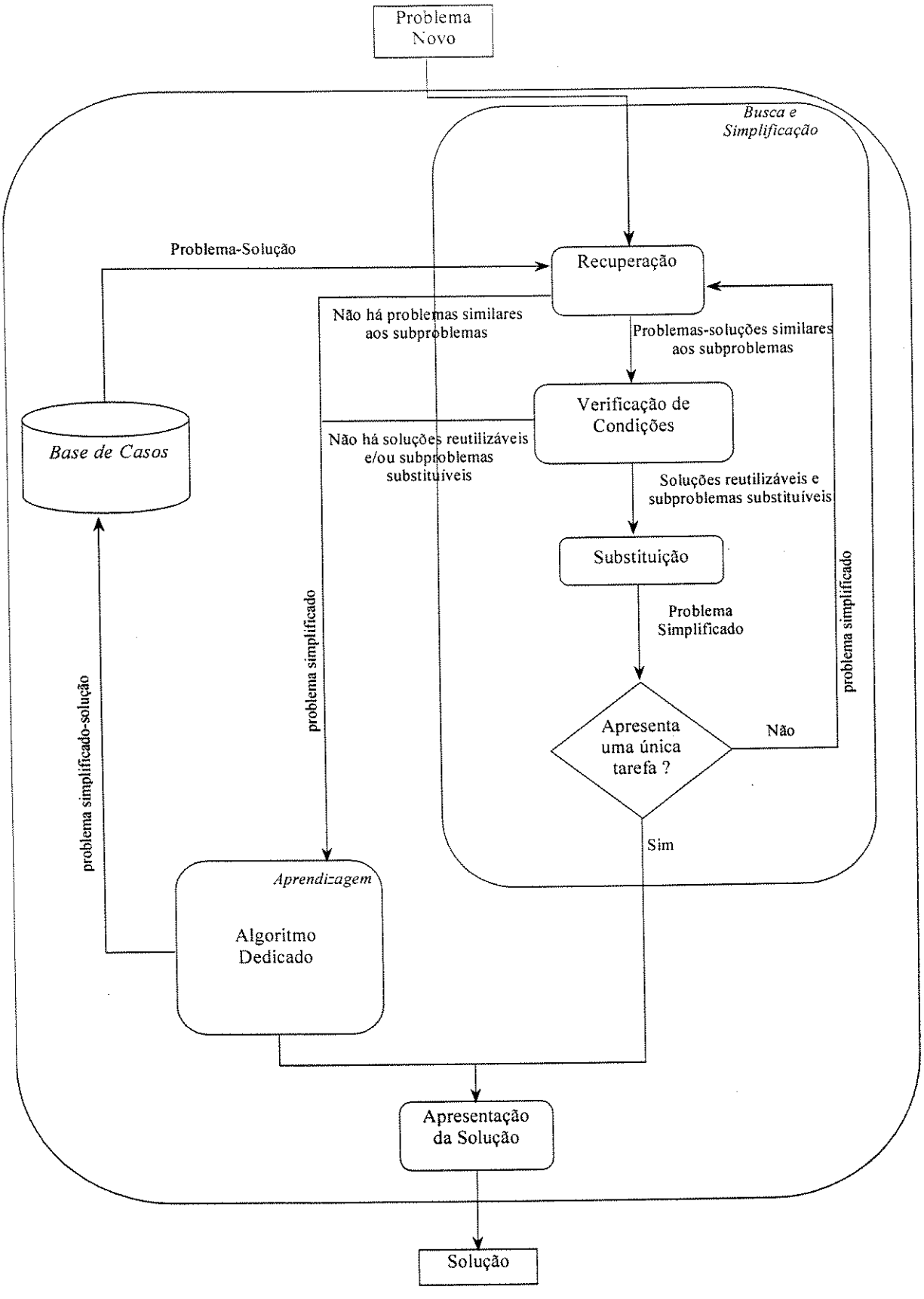



***Figura 10*** *Arquitetura do SEBC*

### *3.3.1 Processo de Busca e Simplificação*

Para que um subproblema do problema novo, similar estruturalmente ao problema armazenado na base de casos, possa ter condições de substituição por um novo processo, primeiramente ele deve obedecer à condição de formação de grupo (seção 3.1). Para que ele seja efetivamente substituído (problema substituível) por um novo processo, as relações de tempo de pronto e prazo de término da Tabela 4 devem ser respeitadas para que a solução recuperada seja reutilizável (possa ser adaptada) na solução deste subproblema. A Figura 11 mostra um subproblema e o seu correspondente na base.

A base de casos pode aprender naturalmente, mas uma estratégia possível para a criação da base inicial é ter problemas armazenados que sejam subproblemas de pelo menos uma parte de problemas novos. Uma possível estratégia para a criação da base de casos inicial com o objetivo Para que novos problemas sejam resolvidos pelo SEBC é importante que os problemas armazenados na base sejam subproblemas de pelo menos uma parte destes problemas novos. Uma prática interessante é ter na base problemas que estatisticamente ocorram com mais freqüência no domínio de uma aplicação. Uma maneira de fazer isto seria gerar uma quantidade $x$ de problemas que fossem escalonáveis e que estivessem dentro de uma determinada classe. Após isto, poderiam ser gerados outros tantos problemas dentro da mesma classe só que com número de tarefas menores, de modo que fossem subproblemas dos outros problemas gerados. Para tanto, deve-se gerar problemas que possuam as características de escalonabilidade e que respeitem à condição de formação de grupo (seção 3.1).

***Tabela 4*** *Relações que devem ser respeitadas na reutilização de um caso*

| Problema Novo | Relação | Problema na Base |
|---|---|---|
| $r_n$ | $\leq$ | $r_b$ |
| $c_n$ | $\leq$ | $c_b$ |
| $d_n$ | $\geq$ | $d_b$ |

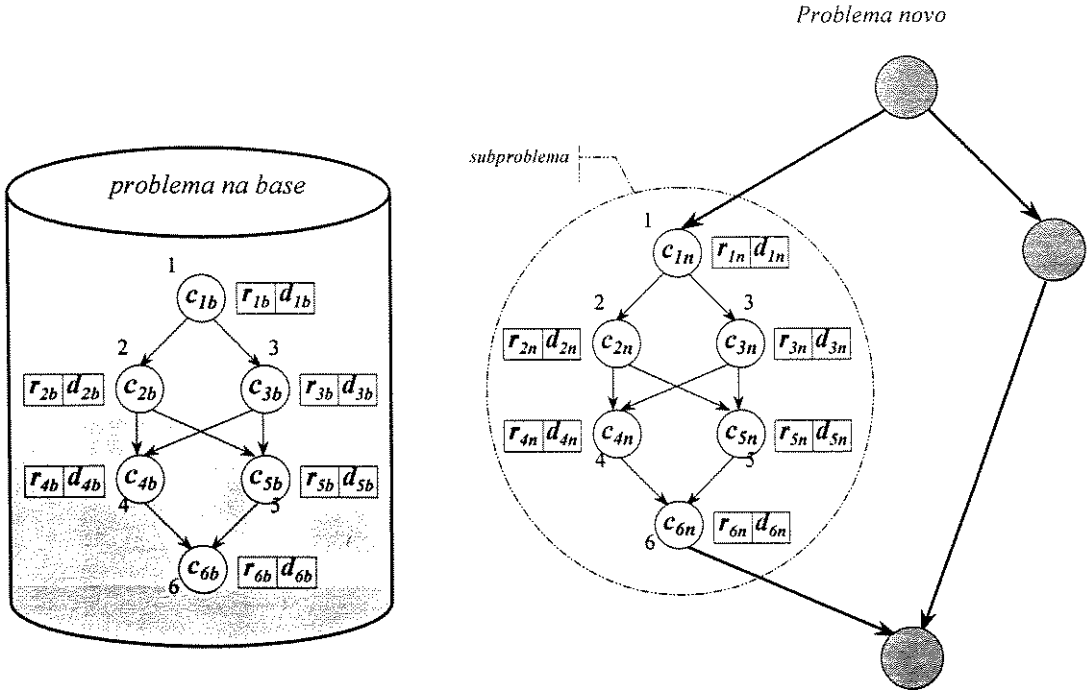



***Figura 11*** *Ilustração de um subproblema e seu similar estruturalmente na base*

O processo de busca e simplificação envolve a recuperação de pares problema-solução, similares aos subproblemas do problema novo apresentado ao SEBC; a verificação das condições de formação de grupo pelo subproblema e das relações de tempo de pronto e prazo de término da Tabela 4; e a substituição do subproblema por um novo processo. Estes passos de recuperação, análise e substituição se repetem até que o problema simplificado apresente uma única tarefa, isto é, tenha sido encontrada a solução com o conhecimento disponível na base de casos ou o fluxo seja desviado para o processo de aprendizagem (item 3.3.2).

Um problema novo é apresentado ao SEBC. Através do GUB procede-se à recuperação. Se não houver problemas similares (como na Figura 12), o fluxo de raciocínio é desviado para o processo de aprendizagem, passando como parâmetro o problema simplificado até o momento do desvio.

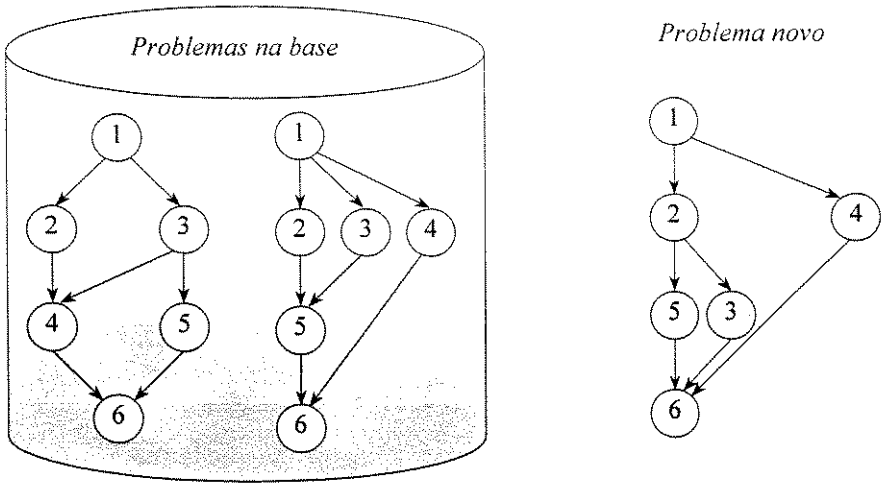



***Figura 12*** *Ausência de problemas similares estruturalmente na base*

Caso haja um ou mais problemas similares estruturalmente na base (como na Figura 13, os problemas 2→4, 3→5 e 1→2) recuperados através do GUB, procede-se à verificação de condições de tempo de pronto e prazo de término de acordo com a Tabela 4.

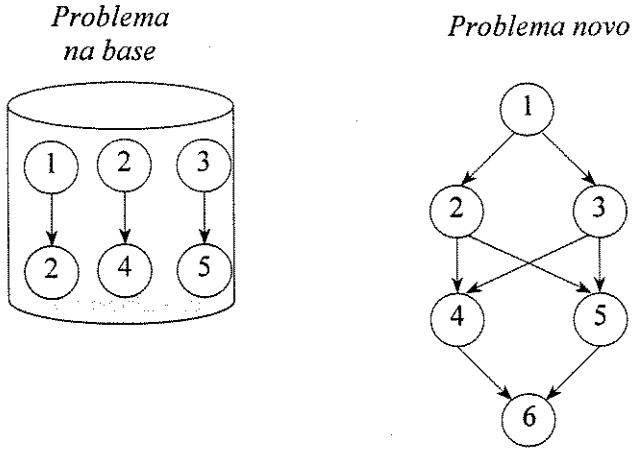



***Figura 13*** *Mais de um problema similar estruturalmente na base*

No passo de verificação de condições, estes problemas similares recuperados são analisados para se verificar se há possibilidade de substituição do subproblema, reutilizando-se a solução recuperada. Duas condições devem ser constatadas: a primeira, se o problema recuperado respeita as condições de tempo de pronto e prazo de término da Tabela 4 e a segunda, se o subproblema do problema maior obedece à restrição de formação de grupo (seção 3.1). Se estas condições não forem satisfeitas (como na Figura 14, onde o prazo de término do subproblema não é maior ou igual ao do problema similar estruturalmente da base e o tempo de pronto do processo de entrada não é menor ou igual ao do problema na base, além de o subproblema não formar um grupo, dado que possui um processo precedente que não possui direta ou indiretamente como precedente o processo de entrada do subproblema),

não é possível proceder à substituição do subproblema por um único processo e continuar o processo de uma nova recuperação até que o problema simplificado apresente uma única tarefa. Nesta situação, o processo de aprendizagem também é chamado e o problema simplificado até o momento é passado como parâmetro.

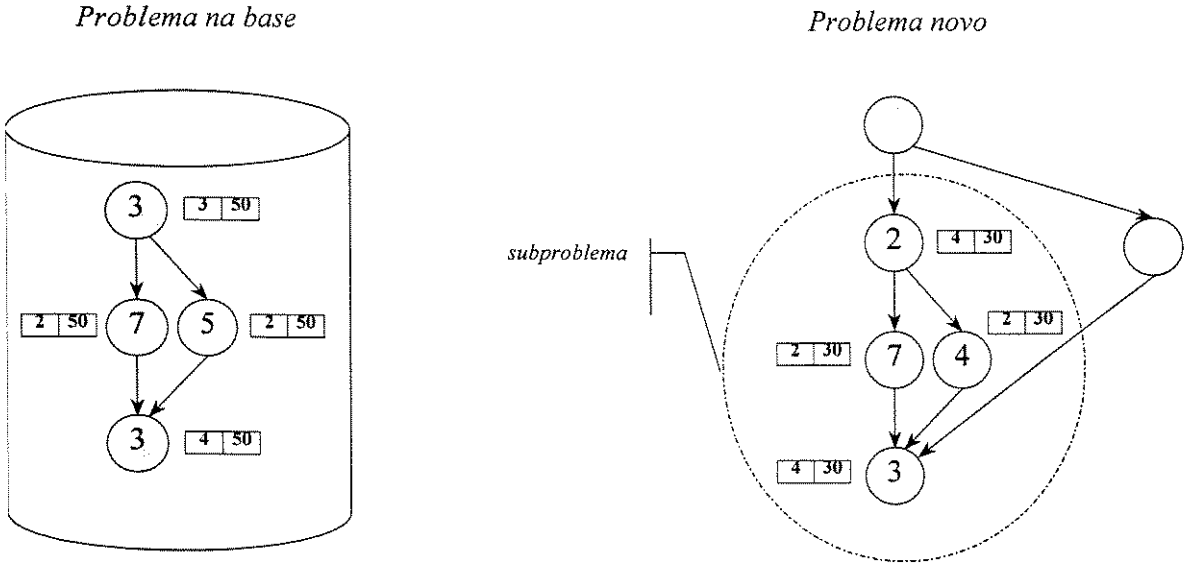



***Figura 14*** *Problema similar estruturalmente na base, mas não atende às restrições de tempo e o subproblema não forma um grupo*

Quando houver a recuperação de problemas similares estruturalmente e soluções reutilizáveis e subproblemas substituíveis (como na Figura 15), o SEBC procede à substituição do subproblema por um novo processo que utiliza a solução adaptada do problema similar recuperado da base (como na Figura 16, onde nota-se também que o cálculo do $c_x$ é "pessimista", isto é, ele pode incluir "vazios" do escalonamento existente para as tarefas $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$ e $\mathbf{T}_4$. Embora isto possa inviabilizar o escalonamento do novo problema, quando isso não ocorrer, os eventuais "vazios" deixados podem ser usados para escalonar processos aperiódicos).

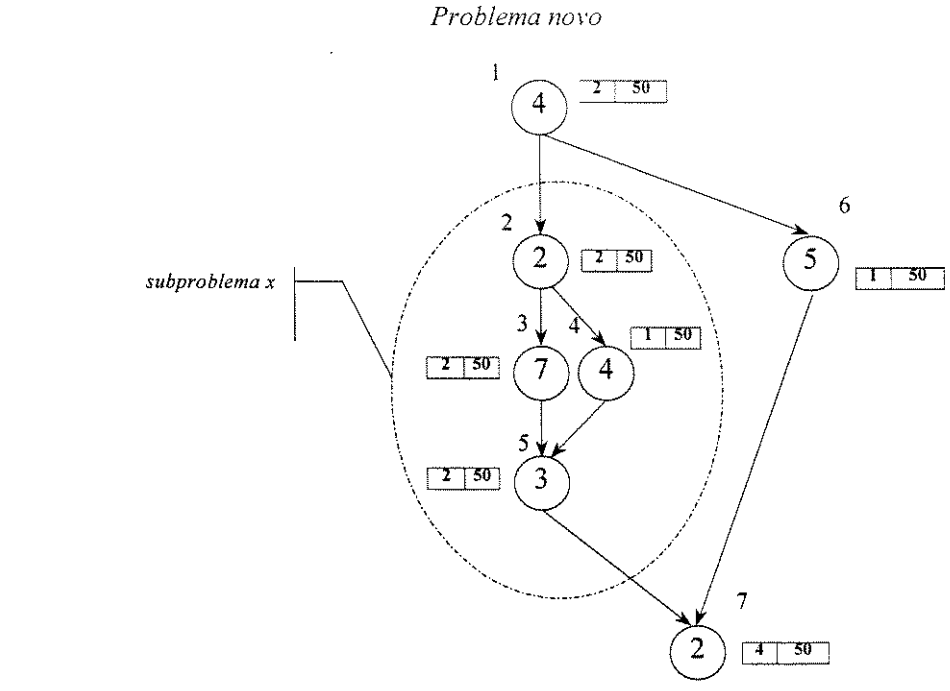
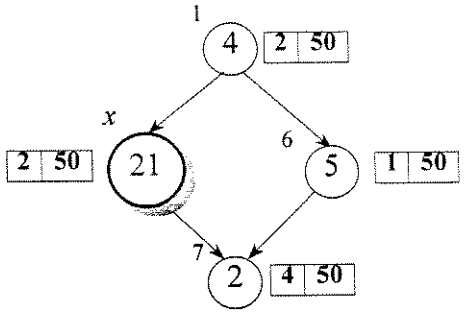



***Figura 15*** *Problema similar estruturalmente na base, atende às restrições de tempo e subproblema forma um grupo*




*Problema Similar Estruturalmente na Base:* $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, $\mathbf{T_4}$
*Subproblema x substituído:* $\mathbf{T_x}$

| | |
|---|---|
| $r_x$= *tempo de pronto do nó de entrada do subproblema* | $r_x = r_1 = 2$ |
| $c_x$= *tempo final – tempo inicial da solução recuperada* | $c_x = t_2(\mathbf{T_4}) - t_1(\mathbf{T_1}) = 21$ |
| $d_x$= *menor* deadline *entre todas as tarefas do subproblema* | $d_x = d_1 = d_2 = d_3 = d_4 = 50$ |

***Figura 16*** *Simplificação do problema através da substituição do subproblema por uma nova tarefa*

Se o problema simplificado possuir apenas uma tarefa, significa que o problema apresentado foi resolvido de forma direta com o conhecimento disponível na base de casos e a solução é apresentada. Se ainda não foi possível resolvê-lo, o problema simplificado é apresentado novamente para o processo de recuperação, verificação de atendimento a condições e substituição até que a solução seja encontrada através da base de casos ou o fluxo seja desviado para o processo de aprendizagem.

### *3.3.2 Processo de Aprendizagem*

Como visto na seção anterior, o processo de busca de problemas similares e simplificação do problema pode terminar sem que o SEBC encontre uma solução para o problema apresentado somente com os casos disponíveis na base até o momento. Quando esta situação ocorre, o problema simplificado até o momento é repassado ao algoritmo dedicado (EPPE) que fornece uma solução. Este problema simplificado e sua solução são em seguida armazenados na base de casos, contemplando o processo de aprendizagem de um novo problema do SEBC (como na Figura 17). Na Figura 18, é mostrado o gráfico de Gantt para a solução final do problema novo da Figura 15.

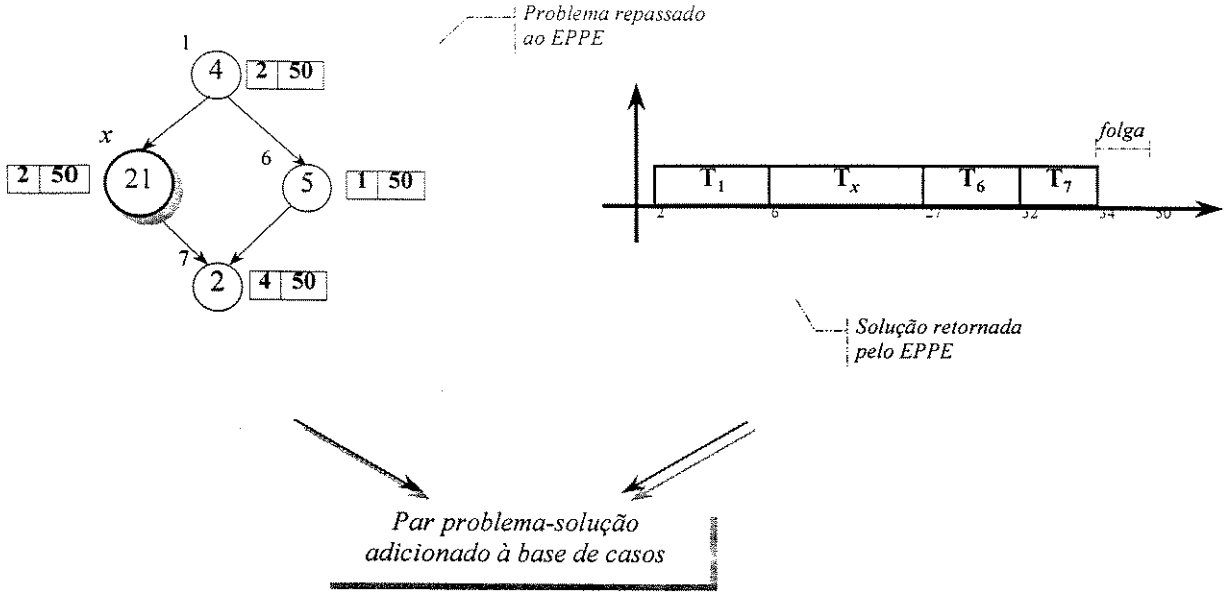



***Figura 17*** *Resolução do problema simplificado pelo EPPE e armazenamento na base de casos*

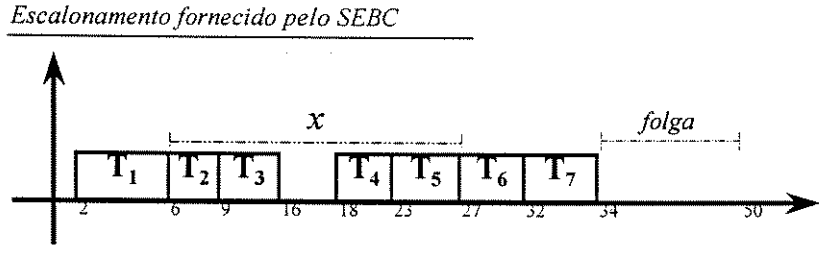



***Figura 18*** *Gráfico de Gantt para a solução final do problema novo*

# 3.4 Conclusões

Este capítulo apresentou a arquitetura de um sistema baseado no método Raciocínio Baseado em Casos (RBC) para a resolução de problemas de escalonamento de processos em um Sistema de Tempo Real Crítico (STRC) monoprocessado (o SEBC, Sistema de Escalonamento Baseado em Casos) e discutiu os principais aspectos de sua implementação.

O problema de escalonamento é representado neste trabalho através de grafos acíclicos dirigidos com atributos. Os vértices do grafo representam os processos, seus rótulos indicam seus tempos de execução e seus atributos trazem os tempos de pronto e prazo de término. Os arcos determinam as relações de precedência.

Um subproblema é uma parte de um problema maior que pode ser resolvida separadamente sem prejuízo para solução do problema como um todo. Os problemas

armazenados na base de casos e os problemas apresentados ao SEBC são representados estruturalmente por meio de grafos acíclicos dirigidos com atributos.

Tem-se a formação de um grupo quando um subproblema é composto por vértices que possuam como precedentes direta ou indiretamente um mesmo vértice de entrada e tenham como sucessores direta ou indiretamente um mesmo vértice de saída. É necessário formar um grupo para que o subproblema possa ser substituído por um único novo processo, que terá como precedente o processo precedente ao vértice de entrada e precederá o processo que o vértice de saída precede, não descaracterizando o problema inicial, isto é, as relações de precedência.

Para se utilizar um problema resolvido anteriormente, armazenado na base de casos, é necessário verificar se ele é similar ao novo problema apresentado ao SEBC. Como estão sendo utilizados grafos para a representação dos problemas, é avaliada a similaridade de grafos, ou seja, a similaridade estrutural. O algoritmo de detecção de isomorfismo de subgrafo proposto por Ullman (1976) e implementado por Messmer (1995) através da GUB (*Graph University of Bern*, uma implementação para casamento de grafo) fornece o mecanismo de recuperação de problemas similares estruturalmente aos subproblemas do problema novo.

O SEBC possui como característica principal a idéia de simplificação de um problema através de substituições de seus subproblemas similares a problemas armazenados em uma base de casos e adaptação das soluções destes problemas na busca de uma solução para o problema apresentado. Para que o subproblema seja efetivamente substituído (problema substituível) por um novo processo, as relações de tempo de pronto e prazo de término da Tabela 4 devem ser respeitadas para que a solução recuperada seja reutilizável (possa ser adaptada) na solução deste subproblema. Nesta implementação, nota-se que o cálculo do tempo de execução do novo processo criado é "pessimista" (exigindo uma demanda maior de processamento), isto é, ele pode incluir "vazios" do escalonamento existente para as tarefas do problema armazenado na base de casos. Embora isto possa inviabilizar o escalonamento do novo problema, quando isso não ocorrer, os eventuais "vazios" deixados podem ser usados para escalonar processos aperiódicos.

Quando o SEBC não encontrar uma solução para o problema apresentado somente com os casos disponíveis na base até o momento de determinada resolução, o algoritmo dedicado (EPPE) é requisitado. O problema simplificado até o momento é repassado ao EPPE que fornece uma solução. Este problema simplificado e sua solução são em seguida armazenados na base de casos, contemplando o processo de aprendizagem de um novo problema do SEBC.

# 4

# Análise Preliminar do SEBC

No capítulo anterior, foi proposto um Sistema de Escalonamento Baseado em Casos, o SEBC, enfocando-se a simplificação de um problema através de substituições de subproblemas similares armazenados em uma base de casos.

Neste capítulo, são propostos nove problemas para serem resolvidos pelo SEBC. O objetivo não é fazer uma análise aprofundada do desempenho do sistema e sim ilustrar a sua utilização e seu potencial através de alguns exemplos de problemas de escalonamento. O desempenho do SEBC para os problemas apresentados é comparado com aquele resultante do uso do Escalonador de Processos Periódicos e Esporádicos, o EPPE, de Melo Jr. (1993).

O capítulo possui a seguinte organização: na seção 4.1, são discutidas as estruturas utilizadas para representação de problemas; na seção 4.2, são descritos os casos da base inicial, ou seja, os problemas e suas respectivas soluções; na seção 4.3, são apresentados os problemas propostos para os experimentos realizados para exemplificar o funcionamento do sistema, juntamente com um detalhamento da resolução de cada um; e, na seção 4.4, é apresentado e discutido o desempenho do SEBC, relativamente ao desempenho do EPPE.

## 4.1 Estruturas dos Problemas

Para cada contexto, os problemas de escalonamento apresentam estruturas que representam suas classes. Portanto, determinadas estruturas tenderão a ocorrer com maior freqüência, variando os valores associados a seus componentes. Essas estruturas podem ser automaticamente aprendidas pelo sistema à medida que problemas lhe são apresentados. No entanto, com o intuito de ilustração (análise preliminar) para este trabalho, foram considerados problemas (armazenados na base de casos e novos, apresentados ao SEBC) formados a partir das estrutruras básicas apresentadas na Figura 19. Conforme introduzido na seção 3.1, os subproblemas de um novo problema a ser resolvido devem formar um grupo. Respeitando-se esta condição, o SEBC trabalha com problemas e subproblemas, gerados automaticamente ou não, com estruturas diversificadas, isto é, relações de precedência e

número de tarefas variados. As estruturas variam de duas a seis tarefas e contemplam uma grande variedade de relações de precedência para cada número de tarefas.

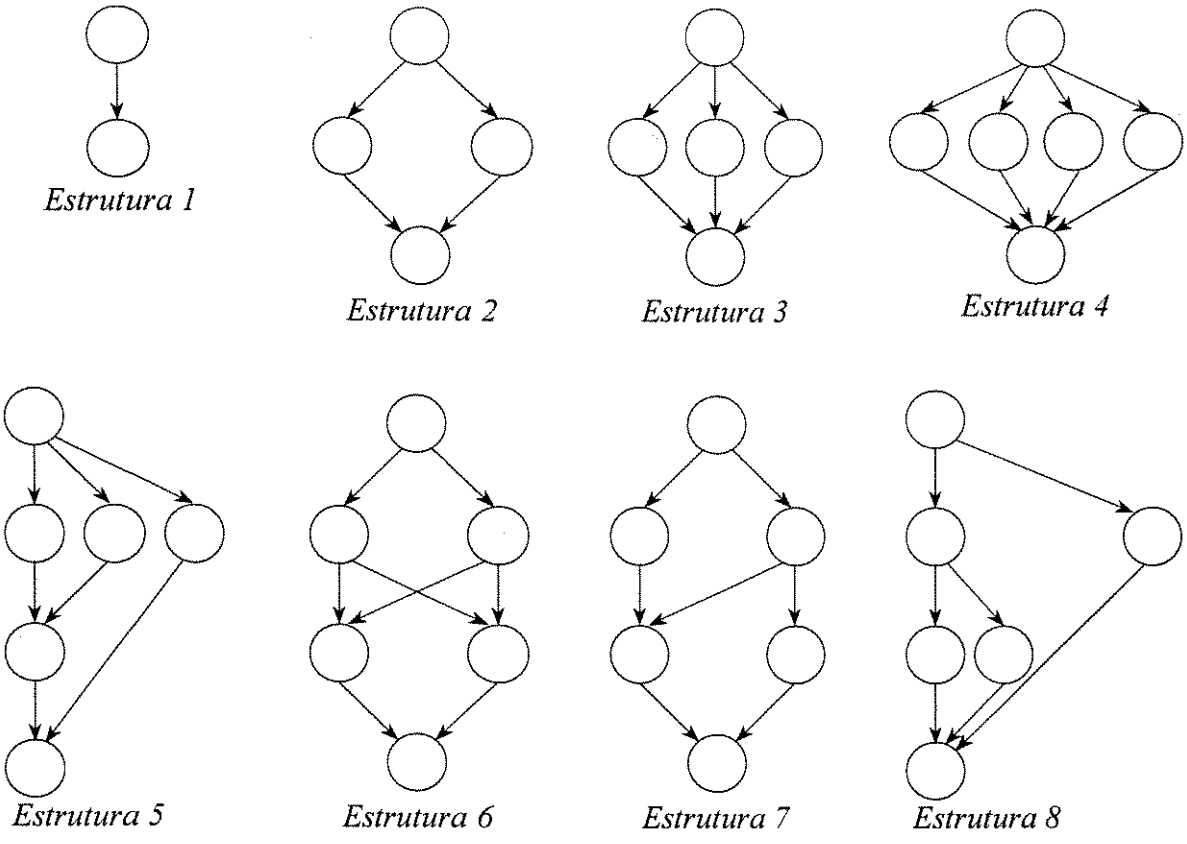



***Figura 19*** *Estruturas básicas dos subproblemas*

## 4.2 Base de Casos

O conhecimento que se tem armazenado na base de casos reflete diretamente no desempenho de um sistema baseado em casos, pois se sua capacidade de encontrar casos passados similares for grande, a probabilidade de se propor soluções para os problemas apresentados ao SEBC aumenta. Uma das características interessantes do RBC, comparativamente a outras abordagens, é que o conhecimento inicial pode ser muito pequeno, podendo o sistema melhorar o seu desempenho à medida que aprende. O processo de aprendizagem de novos conhecimentos também depende deste conhecimento inicial, tanto em termos de qualidade quanto de quantidade. Tendo casos que ocorrem com certa freqüência, torna-se possível resolver mais problemas com o conhecimento disponível até o momento e a necessidade de se utilizar o algoritmo dedicado só aparece quando realmente estes problemas forem muito diferentes dos já aprendidos pelo SEBC.

Foram criados problemas de escalonamento, a partir das estruturas da Figura 19, com características de tempo de pronto, tempo de execução e prazo de término com grande potencial para constituir subproblemas de novos problemas. Eles foram submetidos ao Escalonador de Processos Periódicos e Esporádicos (EPPE) de Melo Jr. (1993), o qual encontrou as respectivas soluções, todas praticáveis. Esses resultados formam os pares problema-solução que constituem a base de casos inicial.

A Figura 20 apresenta um exemplo completo de um caso na base, gerado pelo EPPE e a Figura 21 ilustra de forma gráfica esse mesmo exemplo. Na solução do problema de escalonamento, exemplificado na Figura 20, o atraso quando negativo representa que houve folga no escalonamento, quando zero significa que o escalonamento foi justo e quando maior que zero, que houve atraso. Os segmentos (tarefas escalonadas) têm seus tempos de início e fim representados por $t_1$ e $t_2$.

As Figuras A.1 a A.8, apresentadas no Anexo A, mostram esses casos (pares problema-solução). Cada problema da base foi submetido ao EPPE, que gerou os resultados dos problemas de escalonamento. Os casos armazenados na base são utilizados na resolução dos problemas propostos na seção 4.3.

*Problema*

*Deadline = 30*
*Número de nós: 4*
*Tarefas:*
*(r c d)*
*3 3 30*
*2 7 30*
*15 5 30*
*3 4 30*
*Relações de precedência:*
*1 2*
*1 3*
*2 4*
*3 4*

*Solução*

*Escalonamento:*
$t_1 = 3$ $t_2 = 6$ *segmento = 1*
$t_1 = 6$ $t_2 = 13$ *segmento = 2*
$t_1 = 15$ $t_2 = 20$ *segmento = 3*
$t_1 = 20$ $t_2 = 24$ *segmento = 4*

*Atraso (lateness): -6*

***Figura 20*** *Exemplo de um caso (par problema-solução) armazenado na base*

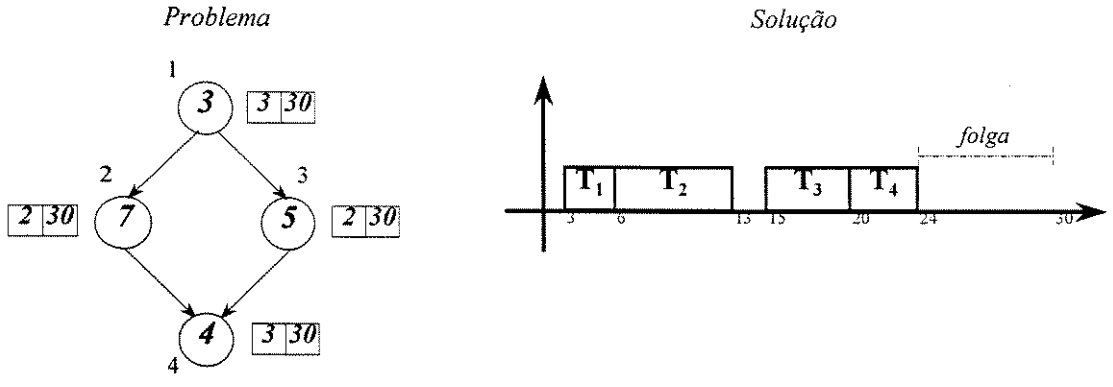



***Figura 21*** *Ilustração gráfica de um caso (par problema-solução)*

# 4.3 Problemas Propostos

Para mostrar a utilização do sistema SEBC, foram resolvidos os problemas mostrados nos itens a seguir, os quais são numerados de $P_1$ a $P_9$ e cujos dados encontram-se representados através de grafos acíclicos dirigidos. Os tempos de execução e de pronto e os prazos de término de cada processo, além de suas relações de precedência foram gerados de maneira que os problemas sejam escalonáveis com o intuito de ilustrar o ganho de desempenho do SEBC sobre o EPPE.

Cada um dos problemas é constituído por subproblemas que podem ser resolvidos separadamente. Para todos os problemas apresentados, é possível chegar a uma solução final através dos casos armazenados na base (seção 4.2), exceto para o problema $P_9$, que, após o processo de adaptação, deve-se utilizar o algoritmo dedicado de escalonamento, implementado através do EPPE. Esta última ação, seguida da incorporação do novo caso, exemplifica a aprendizagem do SEBC. Estes problemas possibilitam ilustrar o desempenho do SEBC em uma situação na qual a base se encontra estável, isto é, seus casos possibilitam adaptações dos problemas propostos até o ponto de se encontrar um caso da base similar estruturalmente e que sua solução possa ser reutilizada para o problema simplificado.

Inicialmente, todos os processos do problema de escalonamento devem ser decompostos em segmentos que constituem a entrada do SEBC e do EPPE. Para se obter estes segmentos, foram decompostas em segmentos todas as instâncias de processos periódicos que ocorrem no intervalo da janela cíclica (o mínimo múltiplo comum entre os períodos dos processos de um problema ainda não segmentado). Esta característica pode ser observada na seção 2.4 do capítulo 2. Os problemas apresentados foram resolvidos inicialmente pelo EPPE. Os processos representados nestes exemplos já se encontram na forma de segmentos. Os itens, a seguir, apresentam com detalhes as resoluções dos problemas propostos através do SEBC e as respectivas soluções finais, além das fornecidas pelo EPPE. São mostrados os grafos para os problemas de escalonamento, juntamente com os tempos de execução e de pronto e os prazos de término de cada processo envolvido, bem como suas relações de precedência.

Adicionalmente, são apresentadas as etapas de simplificação sofridas durante a resolução dos problemas e os resultados finais de escalonamento obtidos.

Para a maioria dos problemas são apresentados dois escalonamentos possíveis, um fornecido pelo SEBC, a partir da simplificação, e o outro fornecido pelo EPPE. Existem algumas diferenças entre os dois resultados. Isto ocorre em função de a política utilizada pelo EPPE escalonar primeiramente a tarefa com maior tempo de execução dentro de um mesmo nível. Duas ou mais tarefas que possuem como precedente uma única tarefa em comum são ditas estarem em um mesmo nível.

### *4.3.1 Problema de Escalonamento $P_1$*

O problema $\mathbf{P_1}$ da Figura 22 possui um subproblema *x*, o qual é similar estruturalmente ao problema 1 da Figura A.1, cuja solução é utilizada para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O problema $\mathbf{P_1}$ tem seu subproblema *x* substituído por um novo processo, obtendo-se a simplificação mostrada através da Figura 23. Este problema simplificado encontra um similar na base de casos (problema 1, Figura A.2), fornecendo uma solução final. Na Figura 24 é mostrado o gráfico de Gantt para a solução final do problema $\mathbf{P_1}$, fornecida pelo SEBC e pelo EPPE.

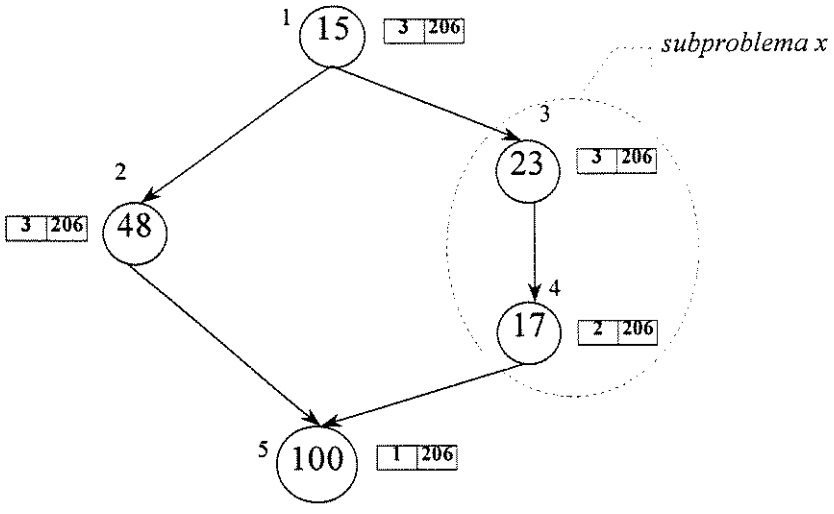



***Figura 22*** *Grafo para o problema de escalonamento P1*

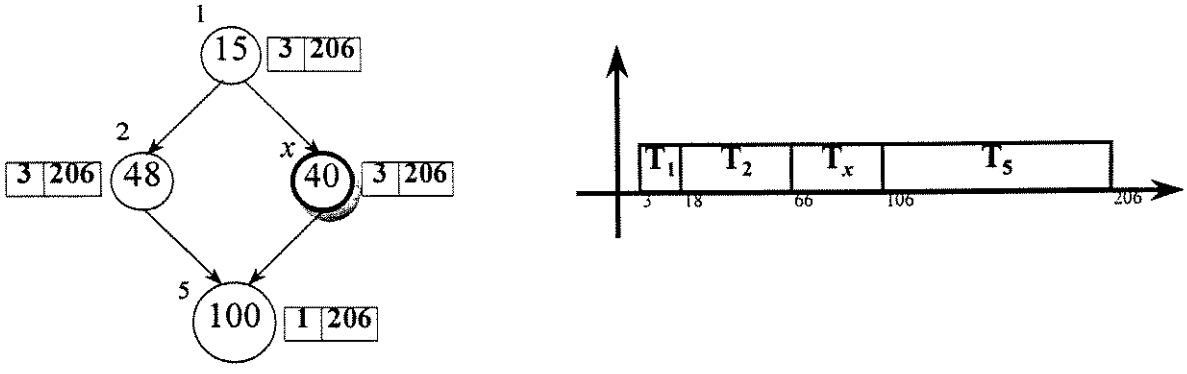



***Figura 23*** *Simplificação do problema P1*

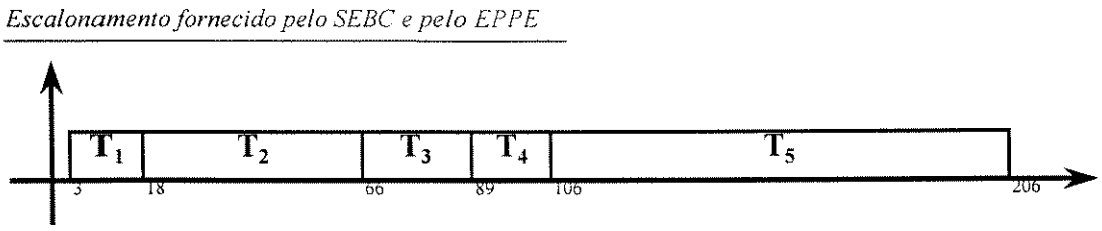



***Figura 24*** *Gráficos de Gantt para a solução final do problema $P_1$*

## *4.3.2 Problema de Escalonamento $P_2$*

O problema $\mathbf{P_2}$ da Figura 25 possui um subproblema *x*, o qual é similar estruturalmente ao problema 1 da Figura A.6, cuja solução é utilizada para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O problema $\mathbf{P_2}$ tem seu subproblema *x* substituído por um novo processo, obtendo-se a simplificação mostrada através da Figura 26. Este problema simplificado encontra um similar na base de casos (problema 4, Figura A.2), fornecendo uma solução final. Na Figura 27, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_2}$, fornecida pelo SEBC e também pelo EPPE.

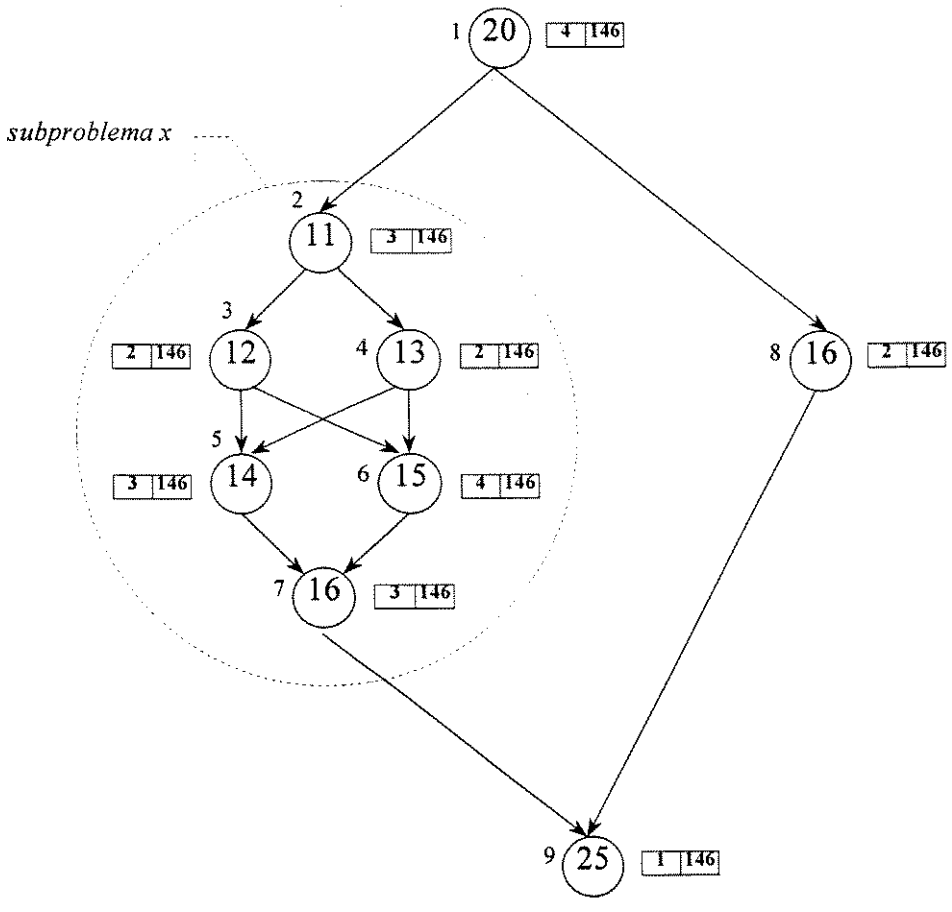



***Figura 25*** *Grafo para o problema de escalonamento $P_2$*

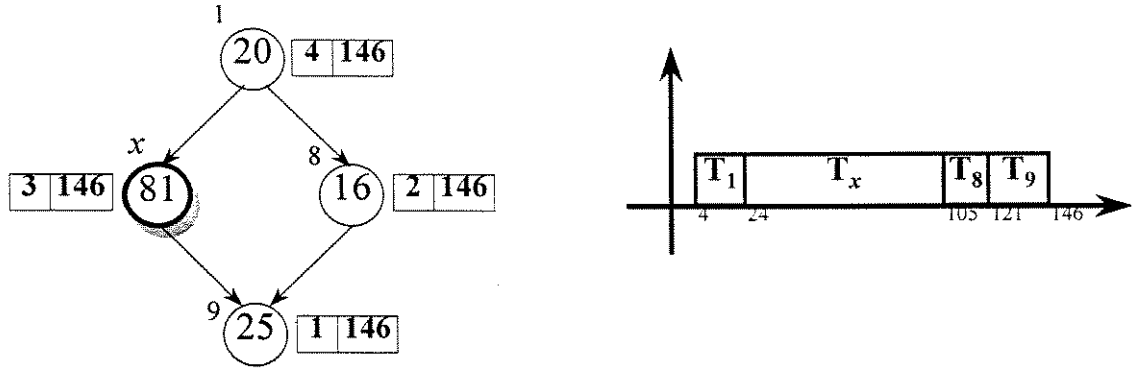



***Figura 26*** *Simplificação do problema $P_2$*

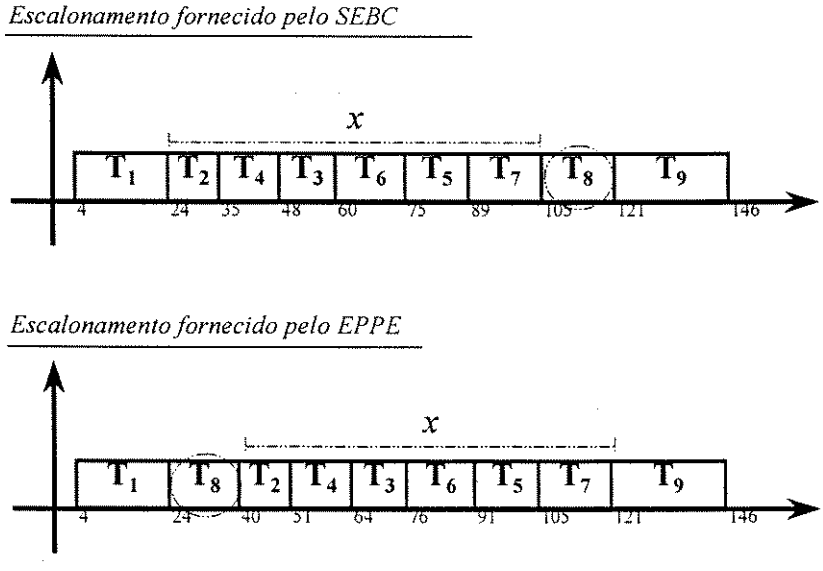



***Figura 27*** *Gráficos de Gantt para a solução final do problema $P_2$*

### *4.3.3 Problema de Escalonamento $P_3$*

O problema $\mathbf{P_3}$ da Figura 28 possui um subproblema *x*, o qual é similar estruturalmente ao problema 1 da Figura A.8, cuja solução é utilizada para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O problema $\mathbf{P_3}$ tem seu subproblema *x* substituído por um novo processo, obtendo-se a simplificação mostrada através da Figura 29. Este problema simplificado encontra um similar na base de casos (problema 3, Figura A.2), fornecendo uma solução final. Na Figura 30 são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_3}$, fornecida pelo SEBC e também pelo EPPE.

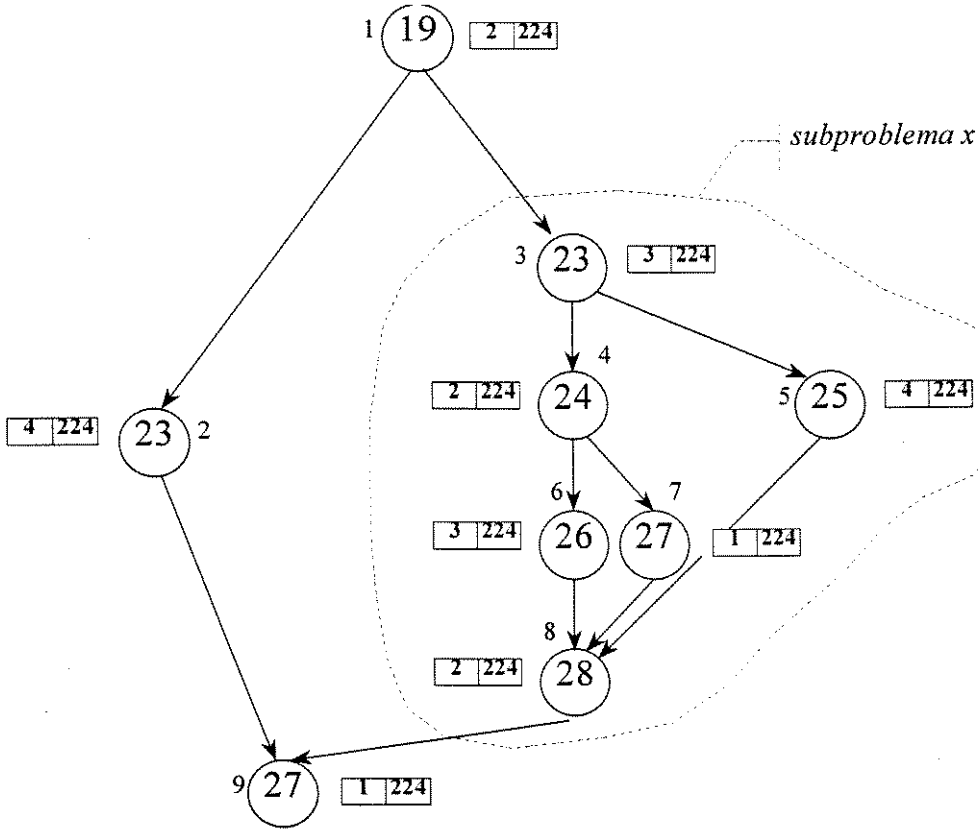



***Figura 28*** *Grafo para o problema de escalonamento P3*

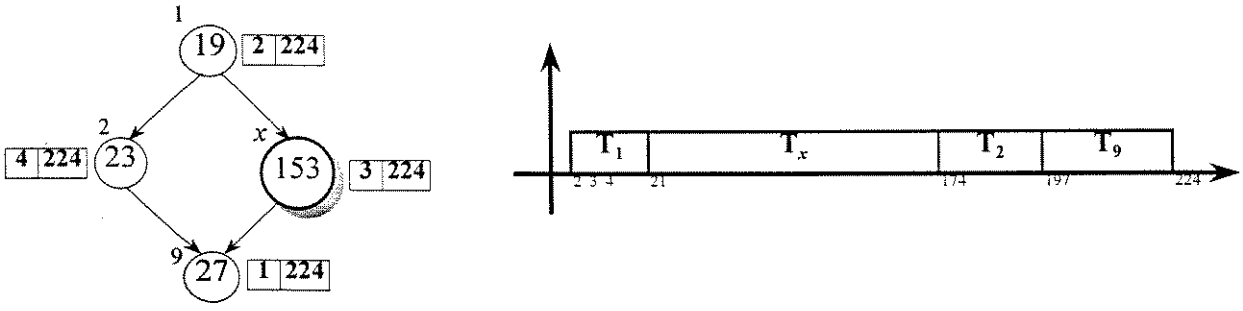



***Figura 29*** *Simplificação do problema P3*

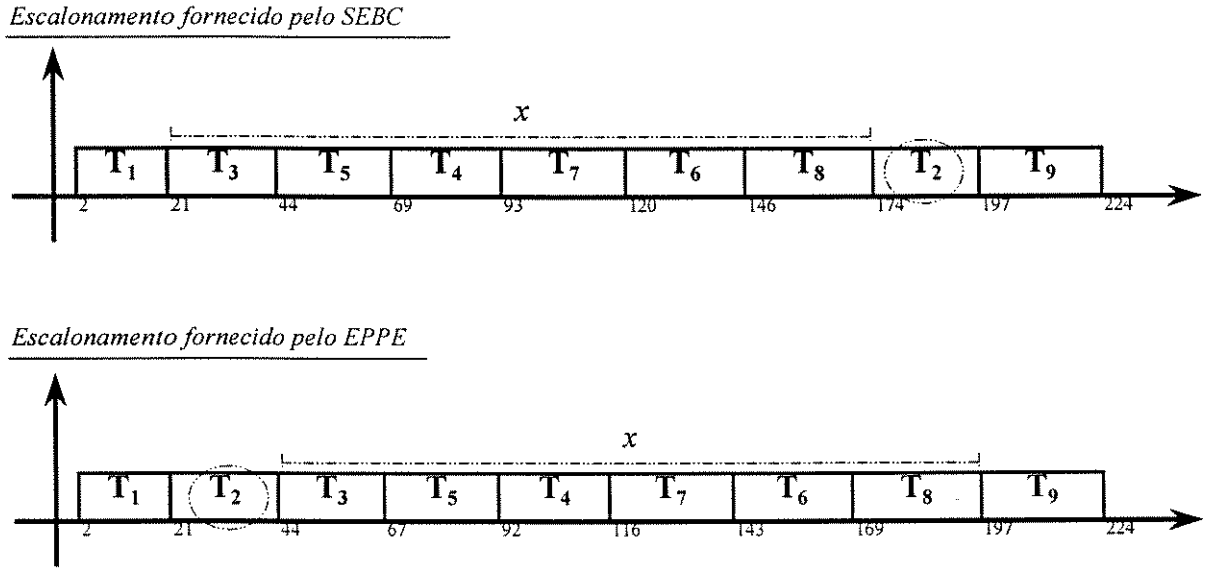



***Figura 30*** *Gráficos de Gantt para a solução final do problema P3*

### *4.3.4 Problema de Escalonamento $P_4$*

O problema $\mathbf{P_4}$ da Figura 31 possui dois subproblemas *x* e *y*, que são similares estruturalmente ao problema 1 da Figura A.4 e ao problema 2 da Figura A.1, respectivamente, cujas soluções são utilizadas para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O problema $\mathbf{P_4}$ tem seus subproblemas *x* e *y* substituídos por novos processos, obtendo-se a simplificação mostrada através da Figura 32. Este problema simplificado encontra um similar na base de casos (problema 2, Figura A.2), fornecendo uma solução final. Na Figura 33 são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_4}$, fornecida pelo SEBC e também pelo EPPE.

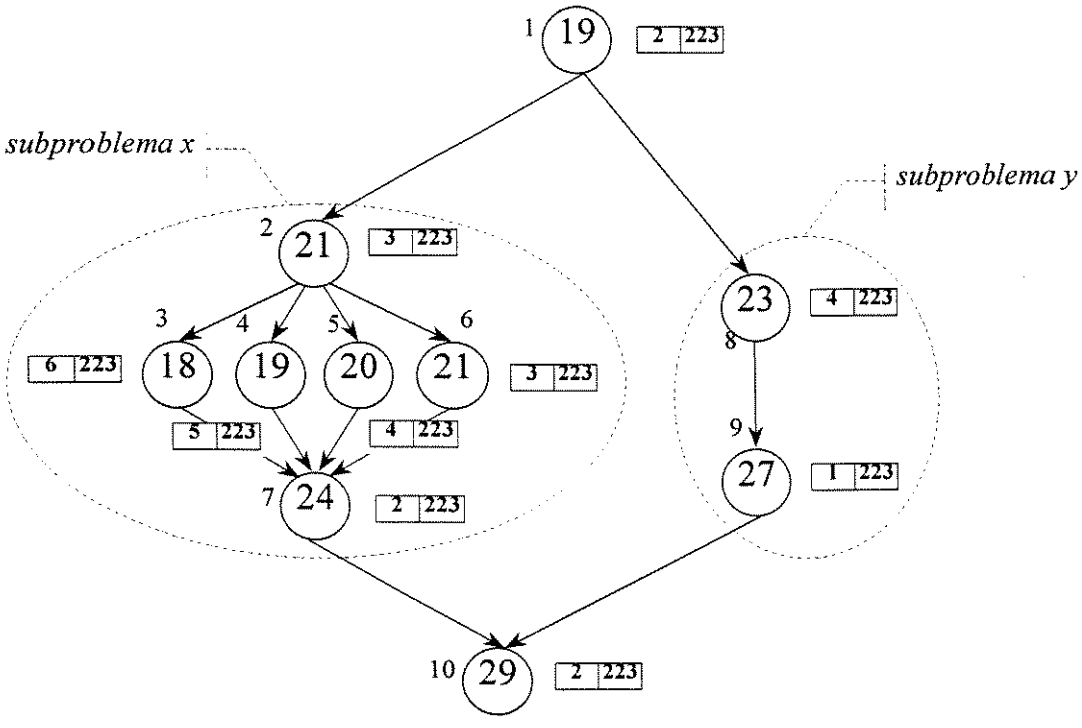



***Figura 31*** *Grafo para o problema de escalonamento $P_4$*

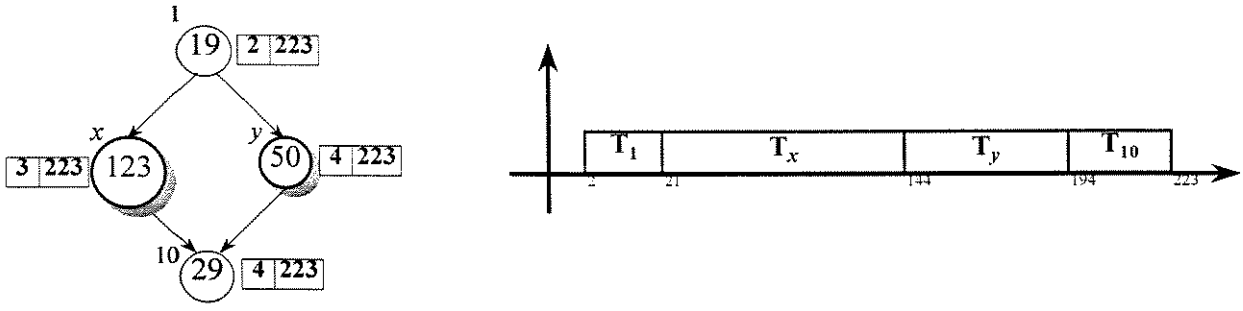



***Figura 32*** *Simplificação do problema $P_4$*

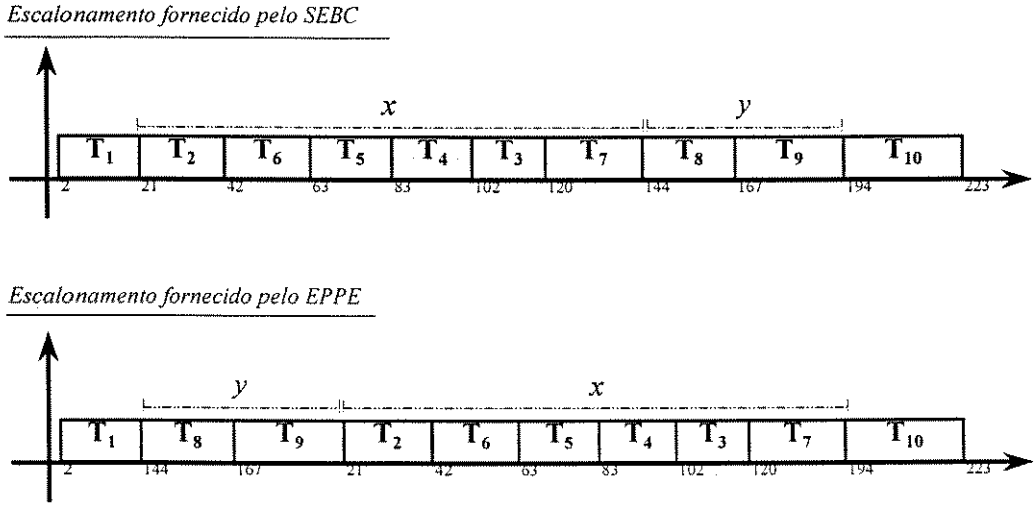



***Figura 33*** *Gráficos de Gantt para a solução final do problema $P_4$*

### *4.3.5 Problema de Escalonamento $P_5$*

O problema $\mathbf{P_5}$ da Figura 34 possui dois subproblemas *x* e *y*, que são similares estruturalmente aos problemas 1 da Figura A.6 e da Figura A.7, respectivamente, cujas soluções são utilizadas para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4 . O problema $\mathbf{P_5}$ tem seus subproblemas *x* e *y* substituídos por novos processos, obtendo-se a simplificação mostrada através da Figura 35. Este problema encontra um similar na base de casos (problema 1, Figura A.3), fornecendo uma solução final. Na Figura 36, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_5}$, fornecida pelo SEBC e também pelo EPPE.

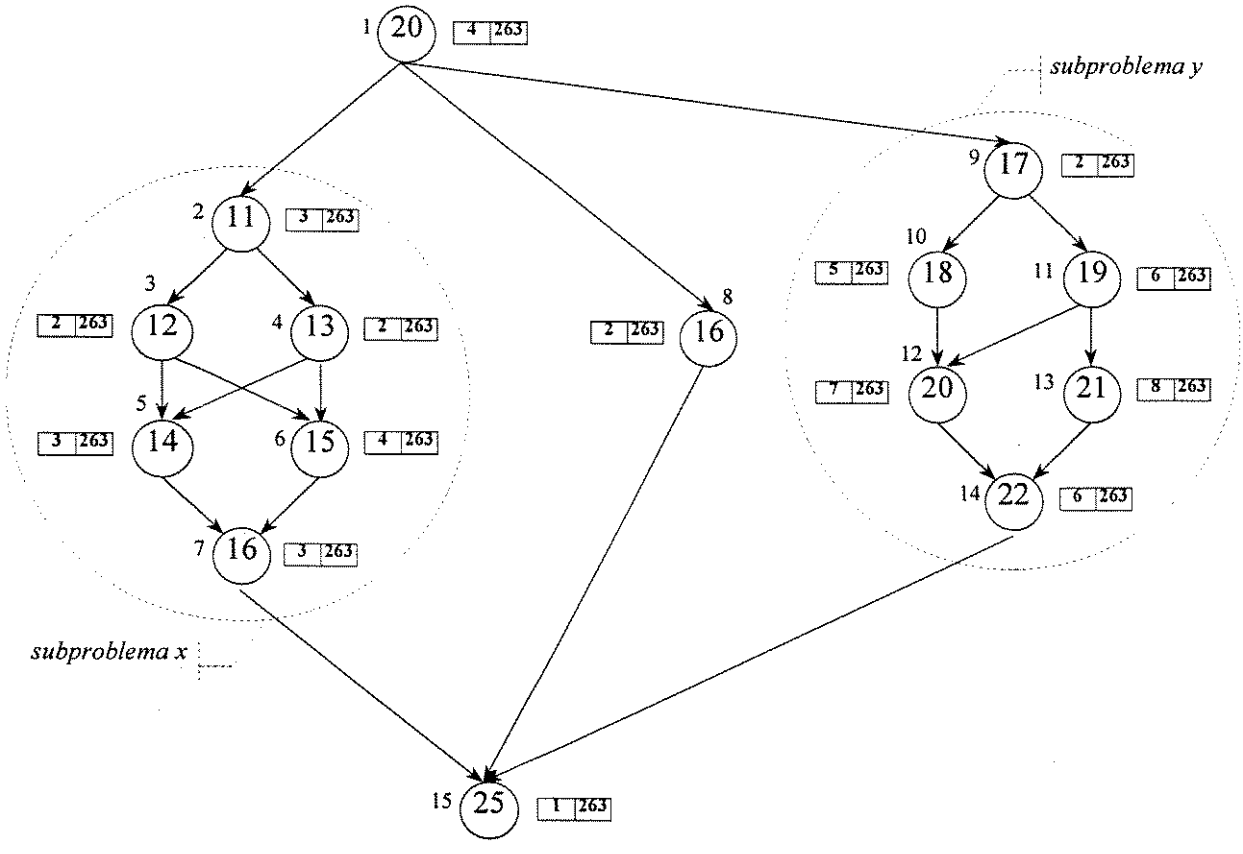



***Figura 34*** *Grafo para o problema de escalonamento $P_5$*

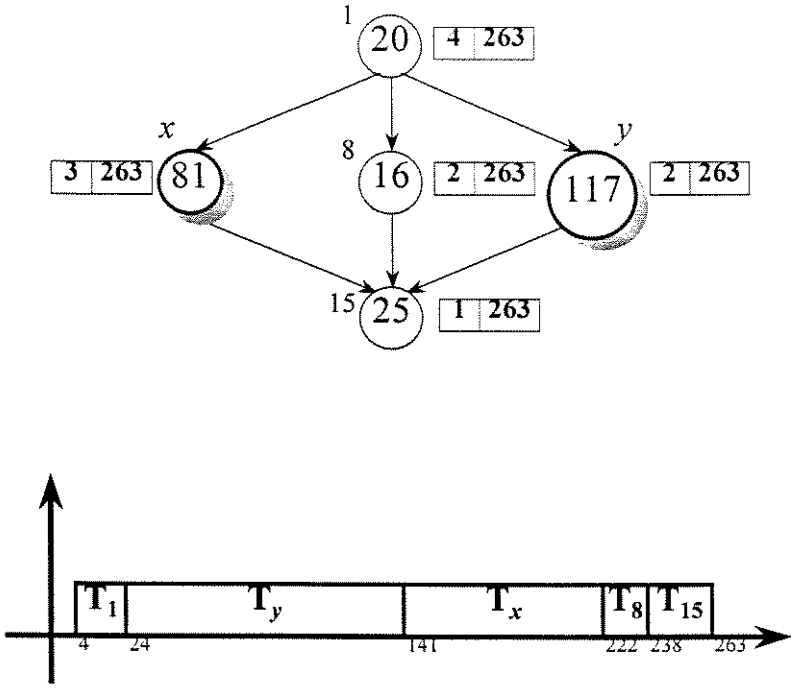
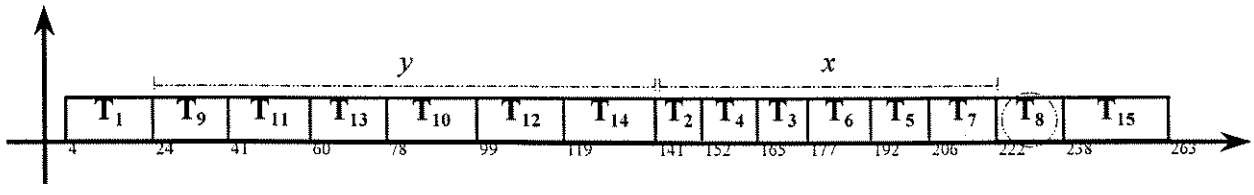



***Figura 35*** *Simplificação do problema $P_5$*

*Escalonamento fornecido pelo SEBC*




*Escalonamento fornecido pelo EPPE*

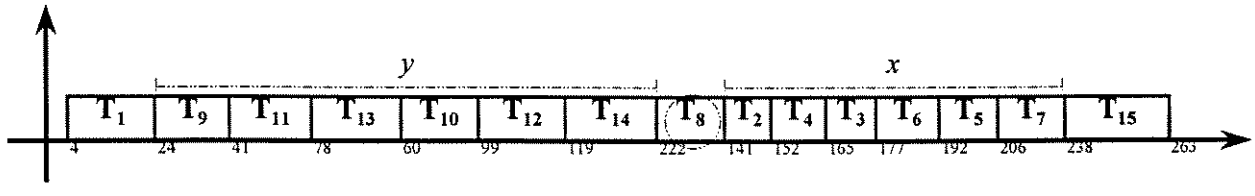



***Figura 36*** *Gráficos de Gantt para a solução final do problema $P_5$*

### *4.3.6 Problema de Escalonamento $P_6$*

O problema $\mathbf{P_6}$ da Figura 37 possui dois subproblemas *x* e *y*, que são similares estruturalmente aos problemas 1 da Figura A.4 e da Figura A.8, respectivamente, cujas soluções são utilizadas para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O problema $\mathbf{P_6}$ tem seus subproblemas *x* e *y* substituídos por novos processos, obtendo-se a simplificação mostrada através da Figura 38. Este problema simplificado encontra um similar na base de casos (problema 1, Figura A.5), fornecendo uma solução final. Na Figura 39, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_6}$, fornecida pelo SEBC e também pelo EPPE.

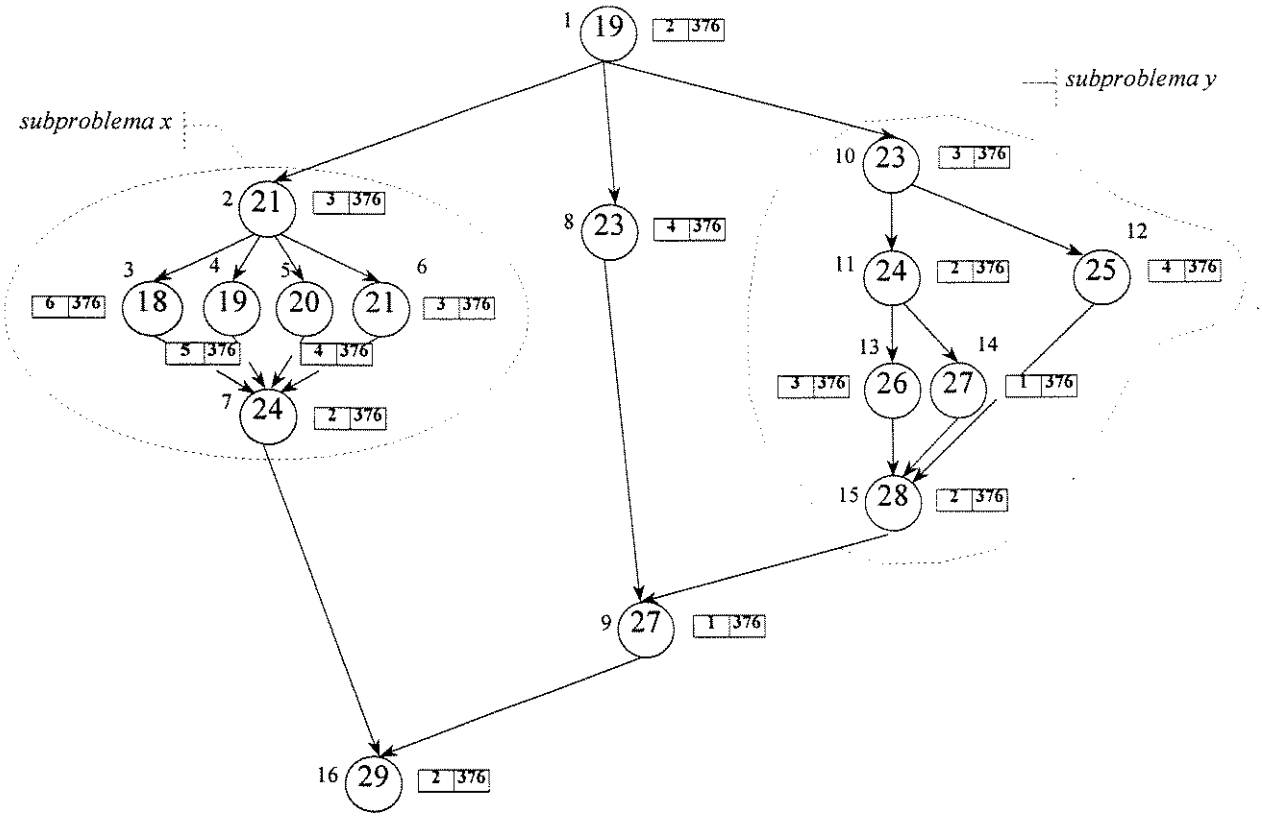



***Figura 37*** *Grafo para o problema de escalonamento $P_6$*

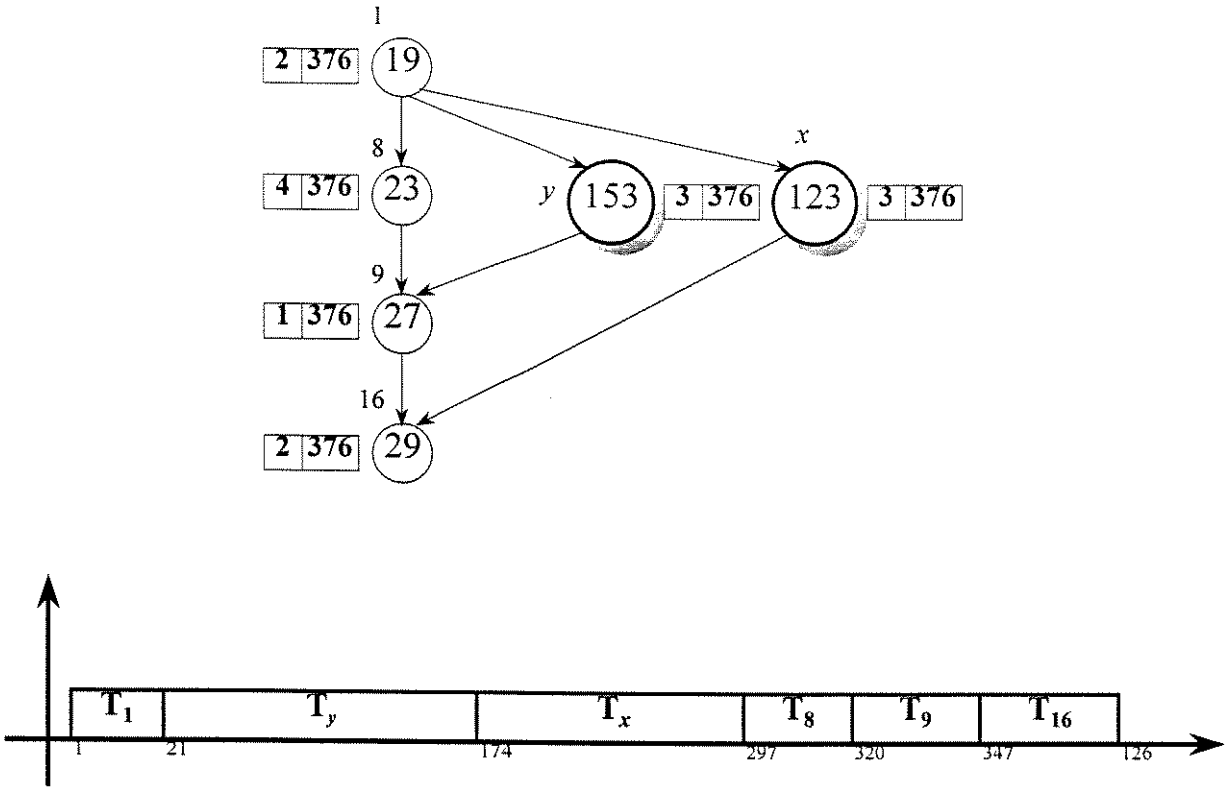



***Figura 38*** *Simplificação do problema $P_6$*

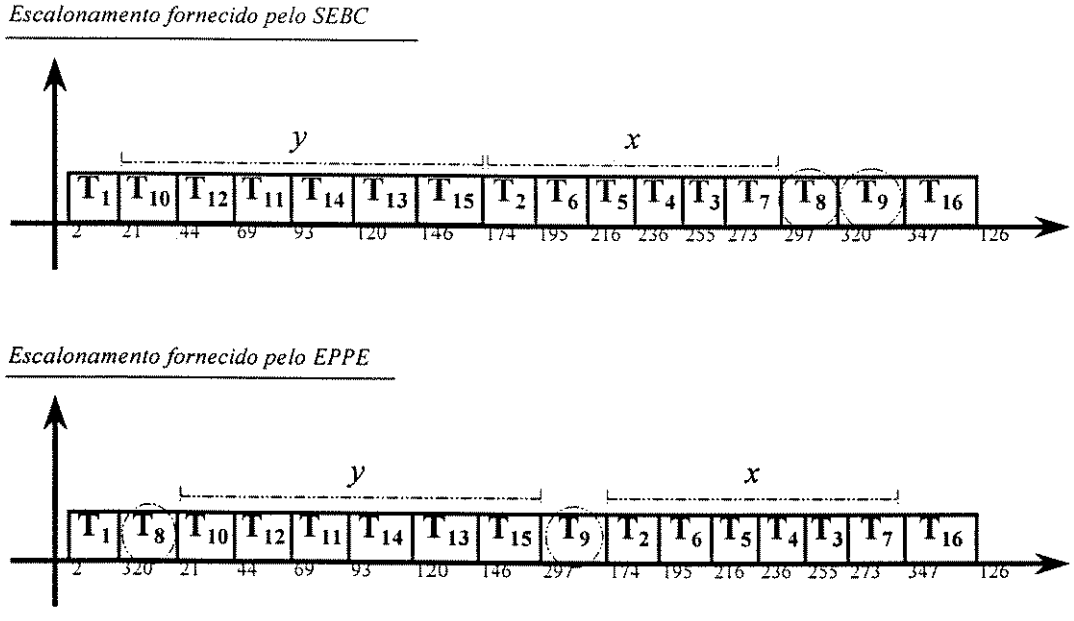



***Figura 39*** *Gráficos de Gantt para a solução final do problema $P_6$*

### *4.3.7 Problema de Escalonamento $P_7$*

O problema $\mathbf{P_7}$ da Figura 40 possui três subproblemas *x*, *y* e *z*, que são similares estruturalmente aos problemas 1 da Figura A.1, da Figura A.4 e da Figura A.8, respectivamente, cujas soluções são utilizadas para a simplificação do problema proposto, uma vez que os casos recuperados atendem às restrições da Tabela 4.

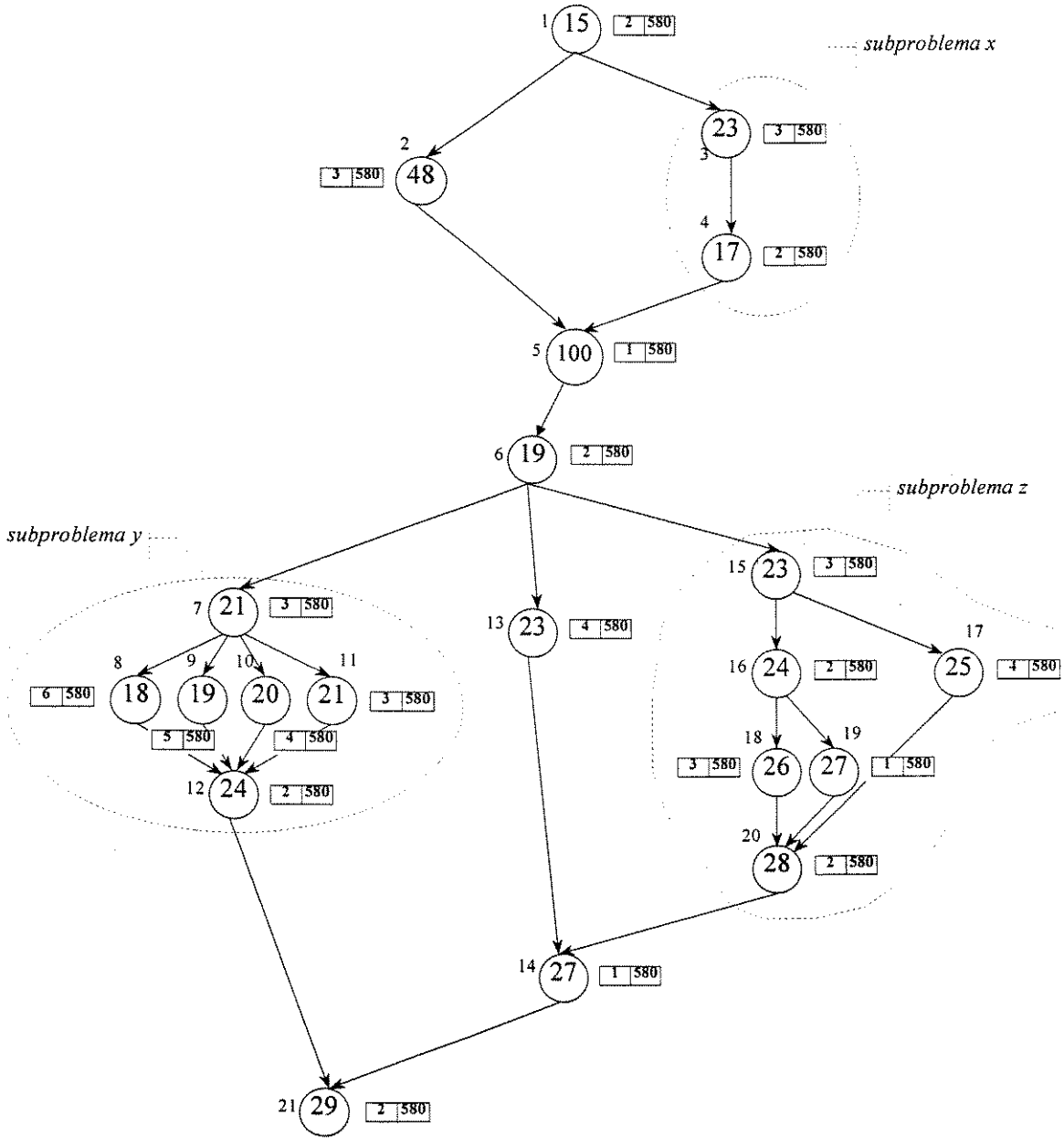



***Figura 40*** *Grafo para o problema de escalonamento $P_7$*

O processo de simplificação para este problema possui duas etapas. Os subproblemas *x*, *y* e *z* são substituídos, dando origem a novos subproblemas *v* e *w*, que são similares estruturalmente aos problemas 1 da Figura A.2 e da Figura A.5, respectivamente. As etapas de simplificação são mostradas através da Figura 41. A simplificação final encontra um similar na base de casos (problema 3, Figura A.1), fornecendo uma solução final. Na Figura 42, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P}_7$, fornecida pelo SEBC e também pelo EPPE.

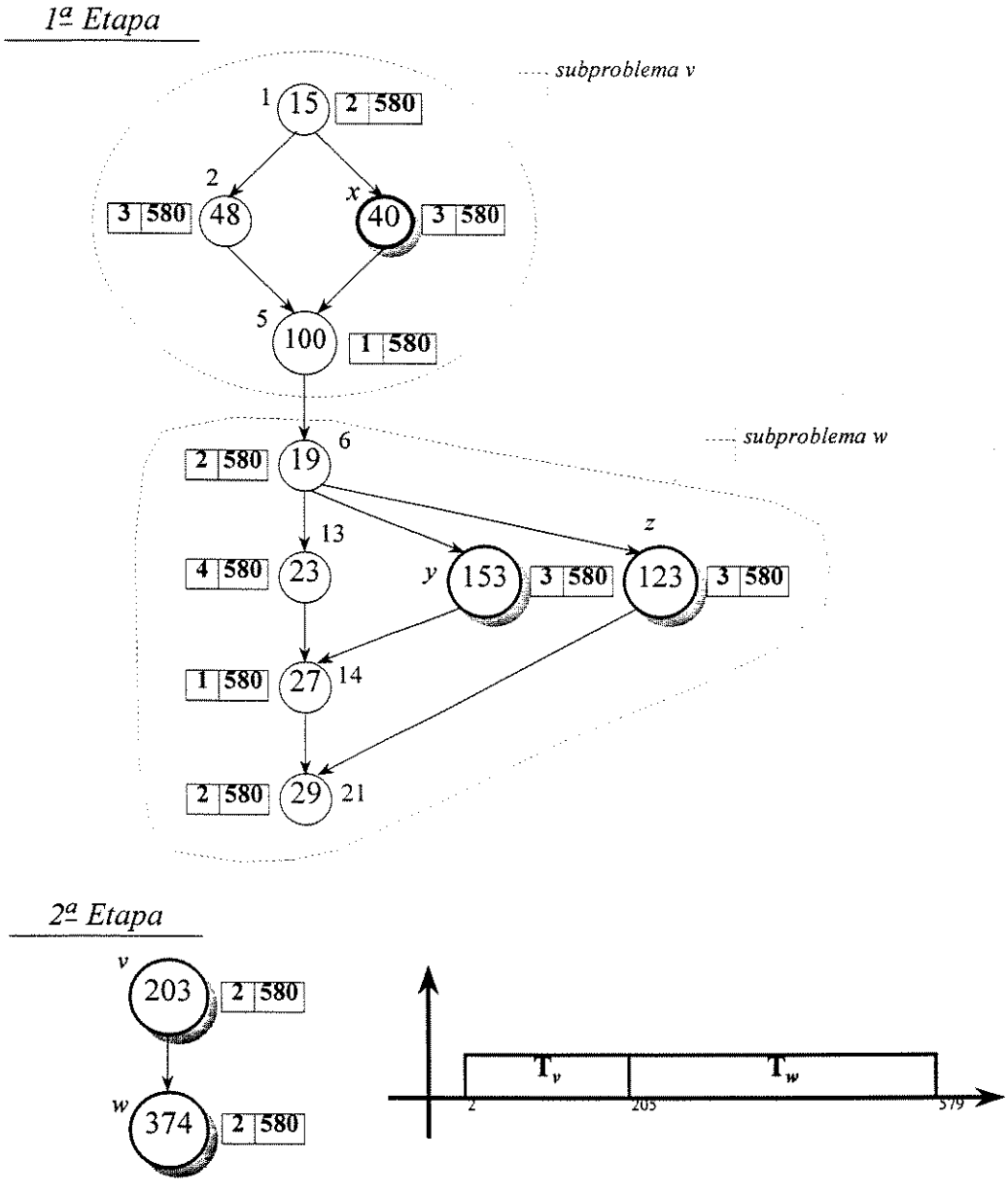



***Figura 41*** *Simplificação do problema P7*

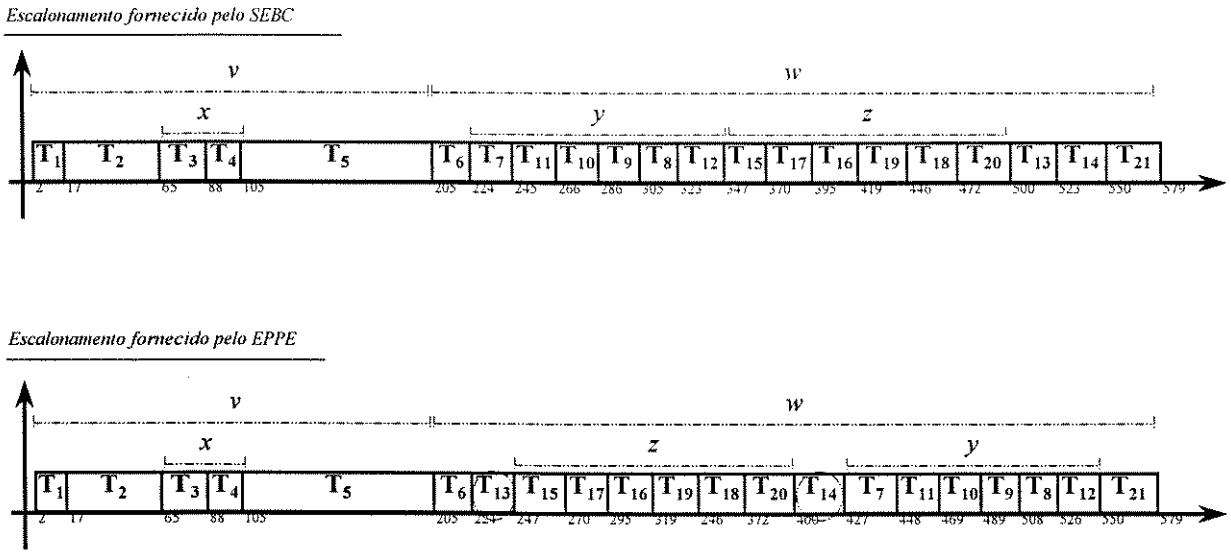



***Figura 42*** *Gráficos de Gantt para a solução final do problema P7*

## *4.3.8 Problema de Escalonamento $P_8$*

O problema $\mathbf{P_8}$ da Figura 43 possui cinco subproblemas *x*, *y*, *z*, *v* e *w*, que são similares estruturalmente aos problemas 1 da Figura A.1, da Figura A.6, da Figura A.7, da Figura A.4 e da Figura A.8, respectivamente, cujas soluções são utilizada para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4. O processo de simplificação para esse problema possui três etapas. Os subproblemas *x*, *y*, *z*, *v* e *w* são substituídos, dando origem a novos subproblemas *a*, *b* e *c*, que são similares estruturalmente aos problemas 1 da Figura A.2, da Figura A.6 e da Figura A.3, respectivamente. Os subproblemas *a* e *b*, por sua vez, são substituídos, dando origem ao novo subproblema *m*, que é similar estruturalmente ao problema 3 da Figura A.1. As etapas de simplificação são mostradas através da Figura 44. A simplificação final encontra um similar na base de casos (problema 5, Figura A.2), fornecendo uma solução final. Na Figura 45, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_8}$, fornecida pelo SEBC e também pelo EPPE.

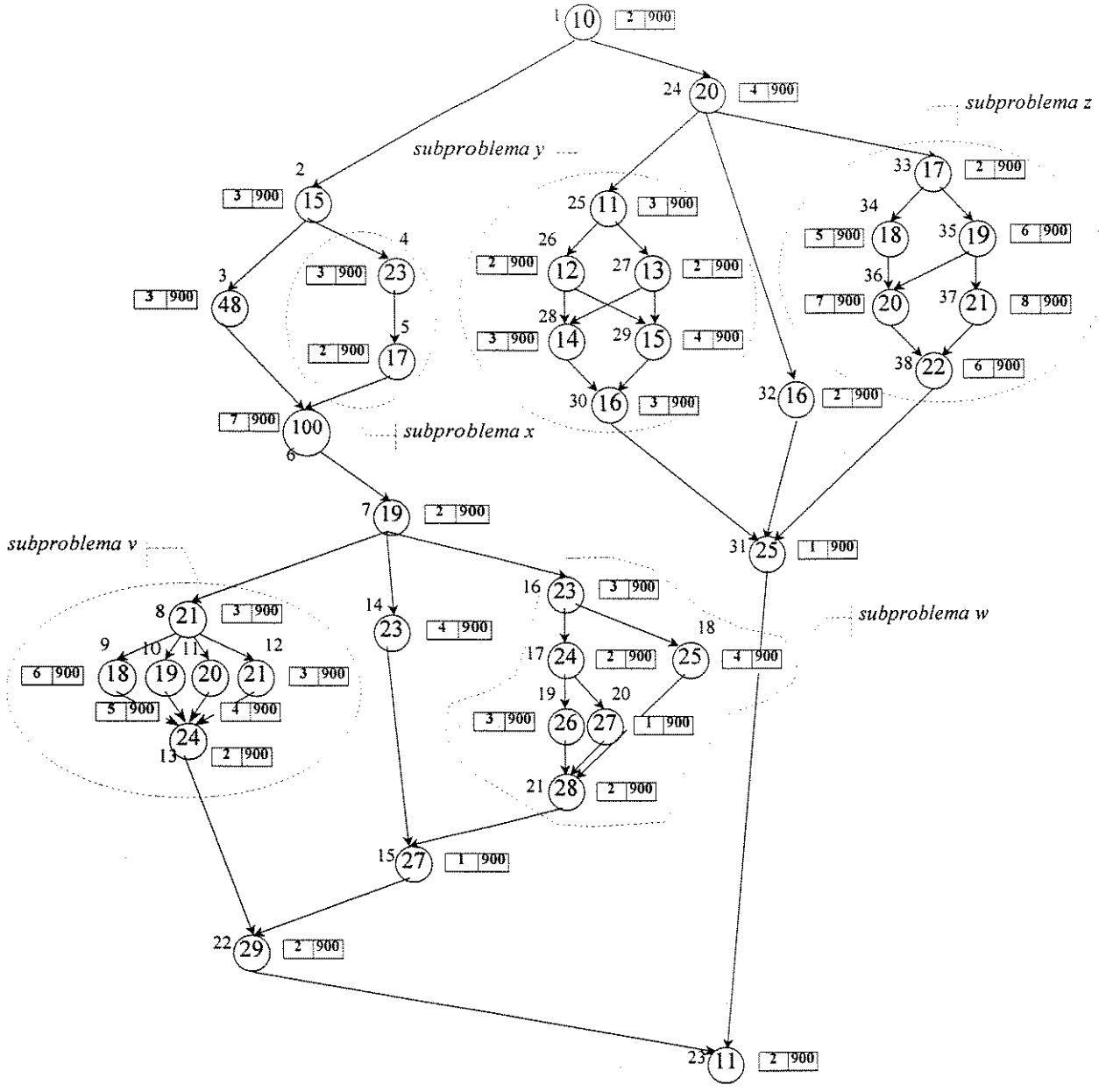
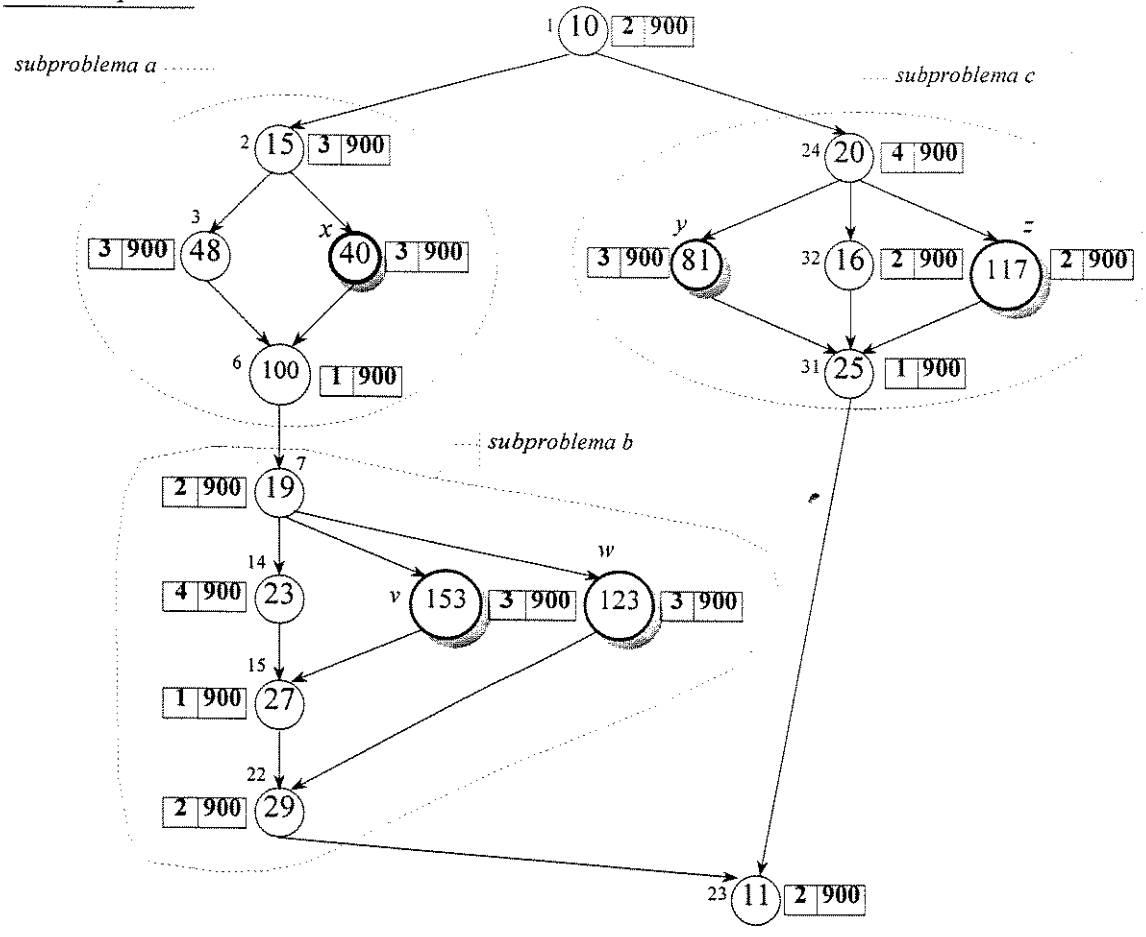
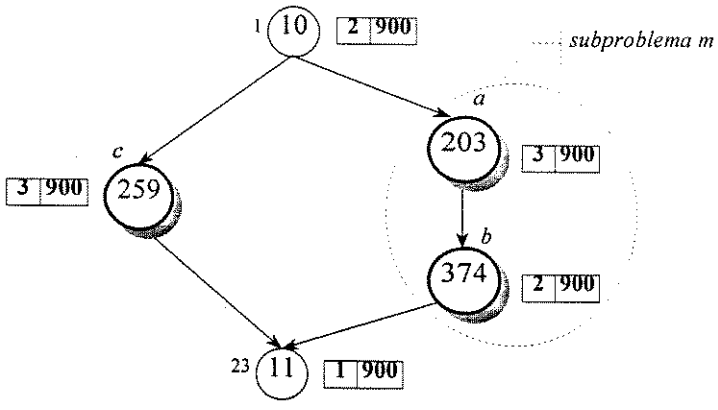



***Figura 43*** *Grafo para o problema de escalonamento P8*

*1ª Etapa*




*2ª Etapa*



*3ª Etapa*

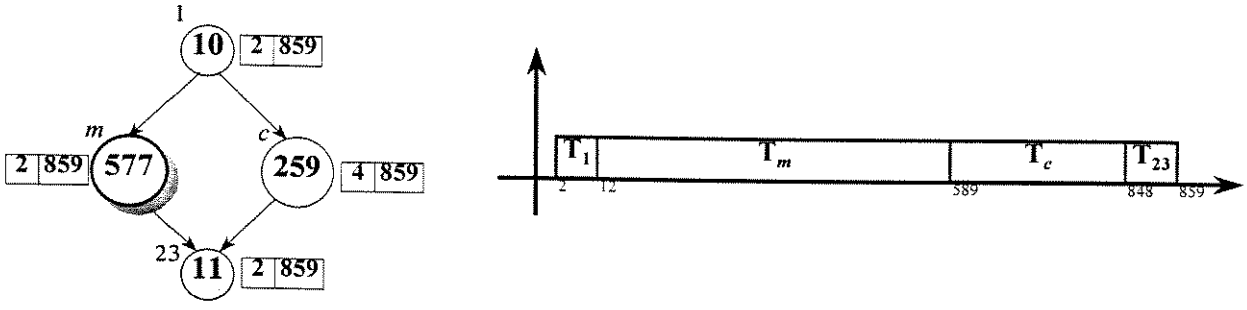
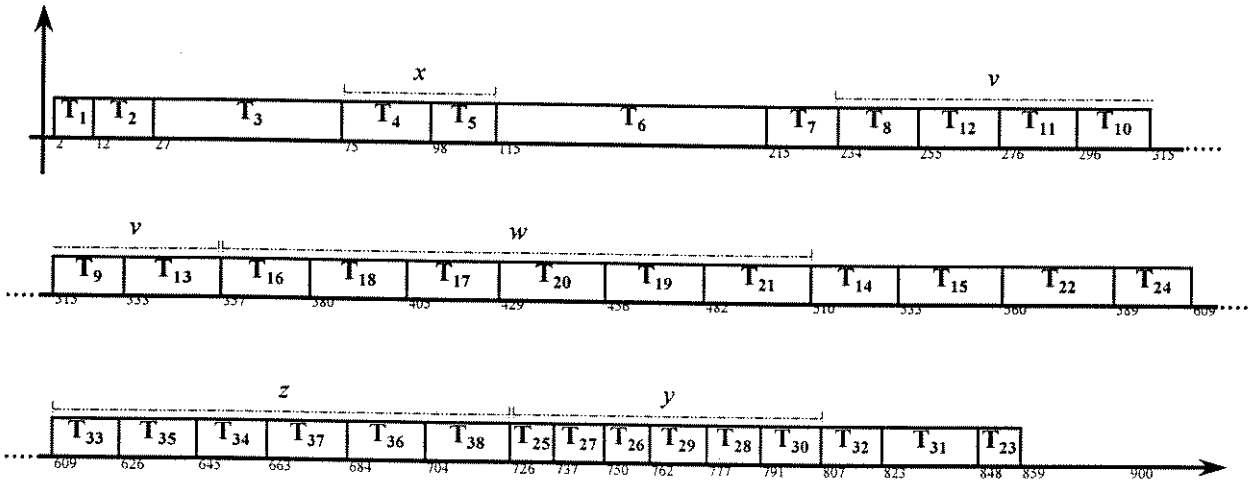



***Figura 44*** *Simplificação do problema P8*

*Escalonamento fornecido pelo SEBC*




*Escalonamento fornecido pelo EPPE*

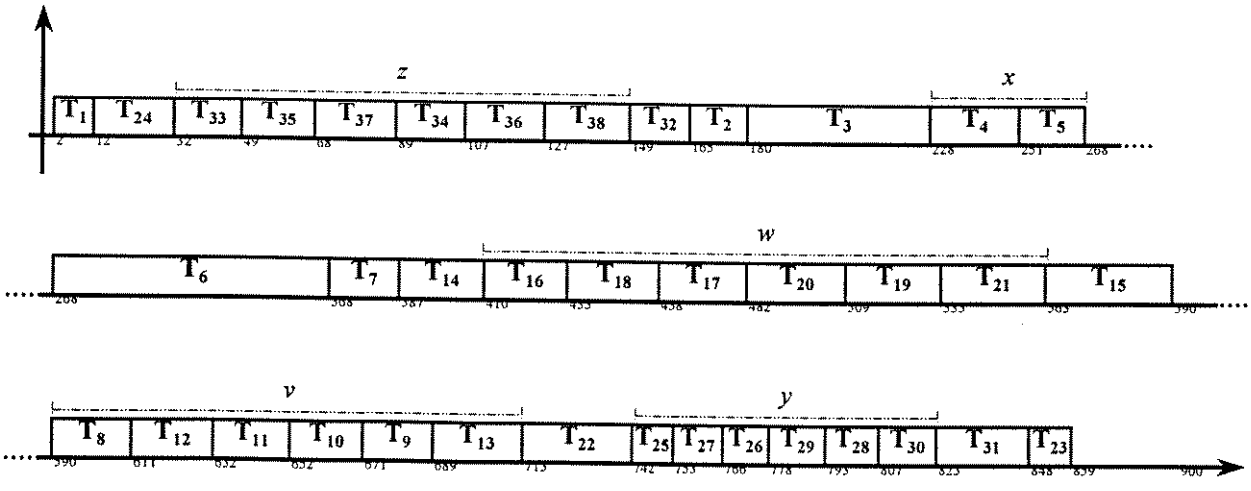



***Figura 45*** *Gráficos de Gantt para a solução final do problema P8*

### 4.3.9 Problema de Escalonamento $P_9$

O problema $\mathbf{P_9}$ da Figura 46 possui três subproblemas *x*, *y* e *z*, que são similares estruturalmente aos problemas 1 da Figura A.1, da Figura A.4 e da Figura A.8, respectivamente, cujas soluções são utilizadas para a simplificação do problema proposto, uma vez que o caso recuperado atende às restrições da Tabela 4.

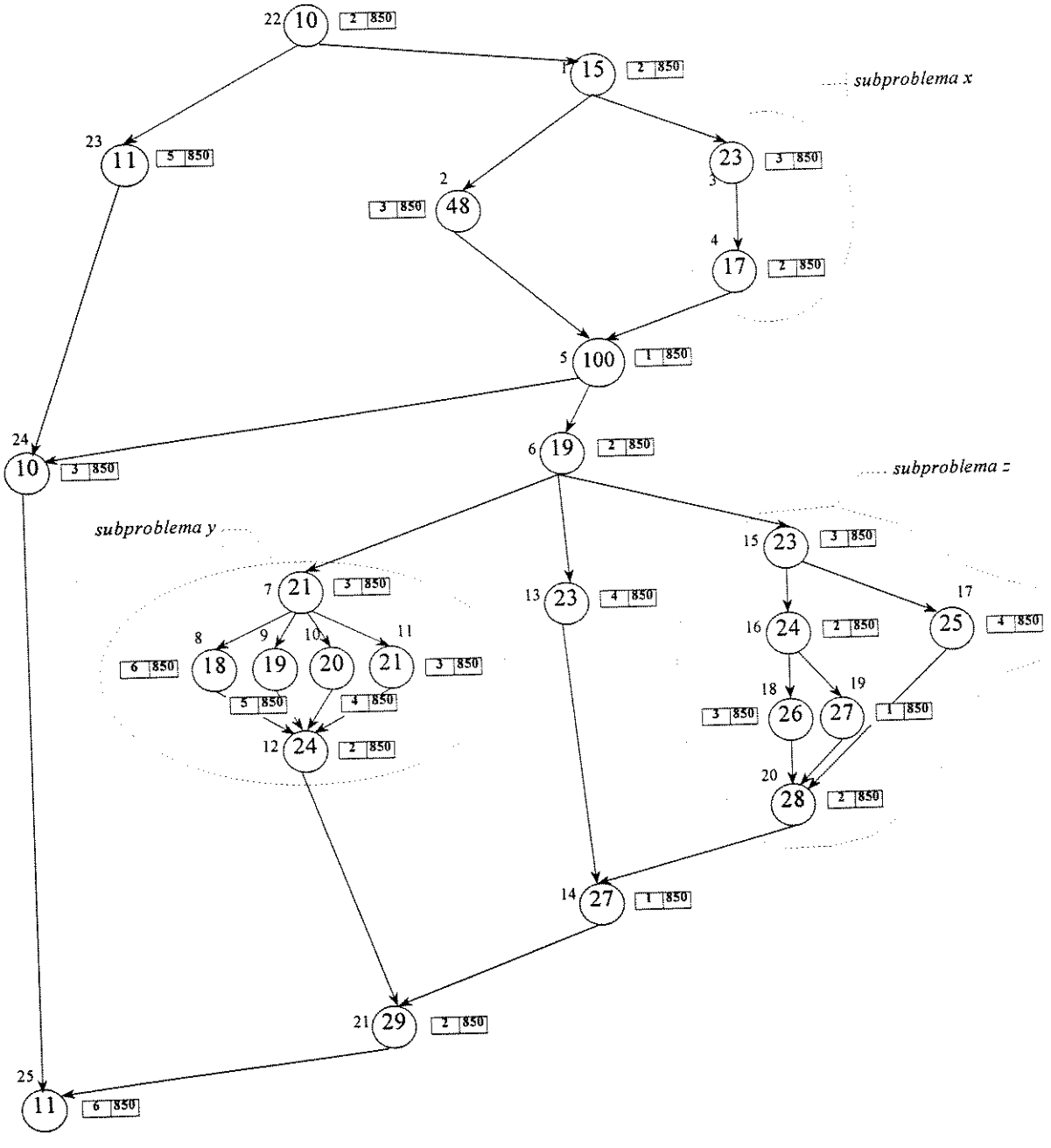
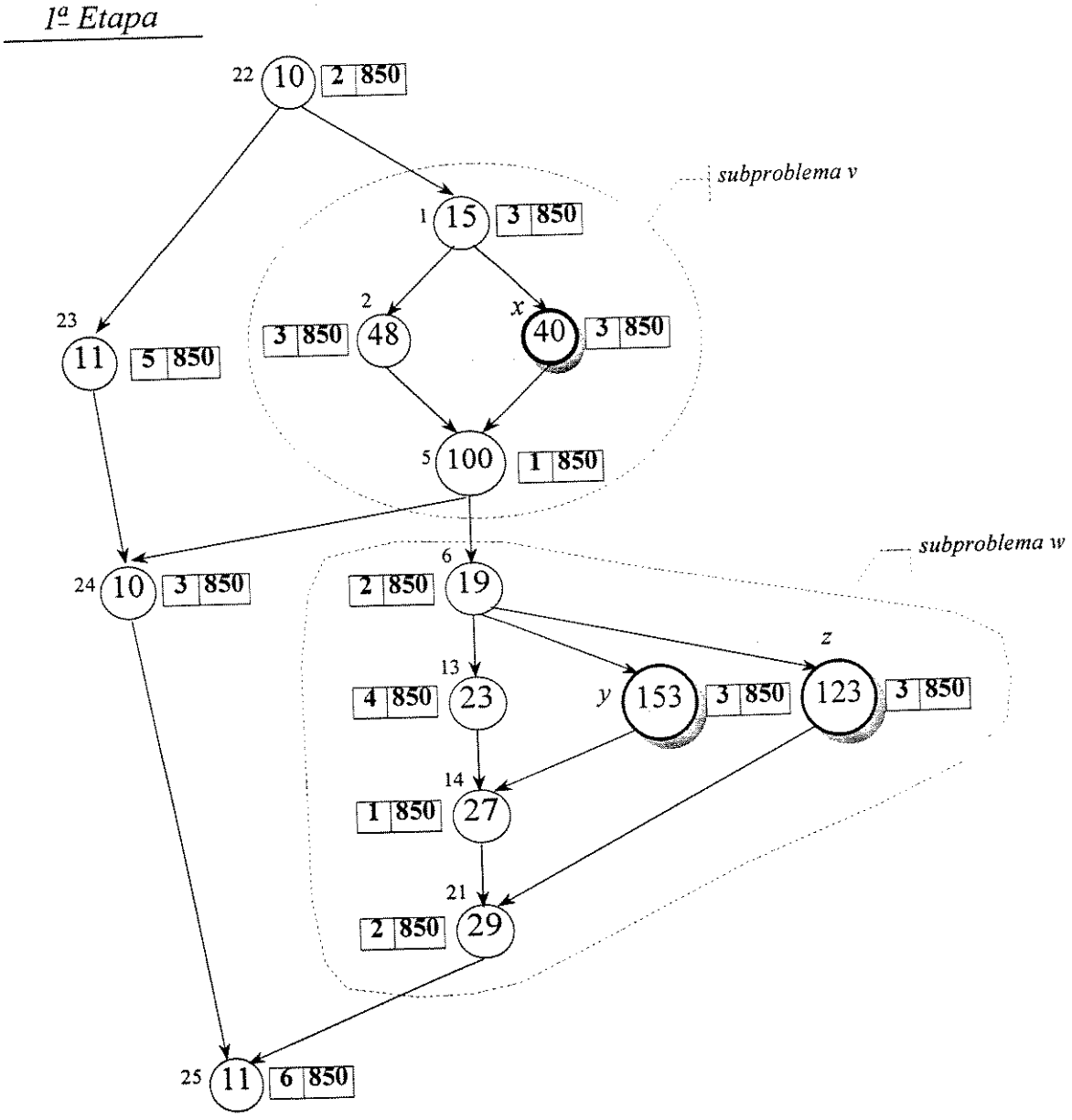



***Figura 46*** *Grafo para o problema de escalonamento $P_9$*

O processo de simplificação para este problema possui duas etapas. Os subproblemas *x*, *y* e *z* são substituídos, dando origem a novos subproblemas *v* e *w*, que são similares estruturalmente aos problemas 1 da Figura A.2 e da Figura A.5, respectivamente. As etapas de simplificação são mostradas através da Figura 47. A simplificação final não encontra um similar na base de casos, e, portanto, o problema simplificado é repassado ao EPPE para ser escalonado. O par problema-solução resultante passa, então, a compor a base de casos, como conseqüência do aprendizado do SEBC. Na Figura 48, são mostrados os gráficos de Gantt para a solução final do problema $\mathbf{P_9}$, fornecida pelo SEBC e também pelo EPPE.



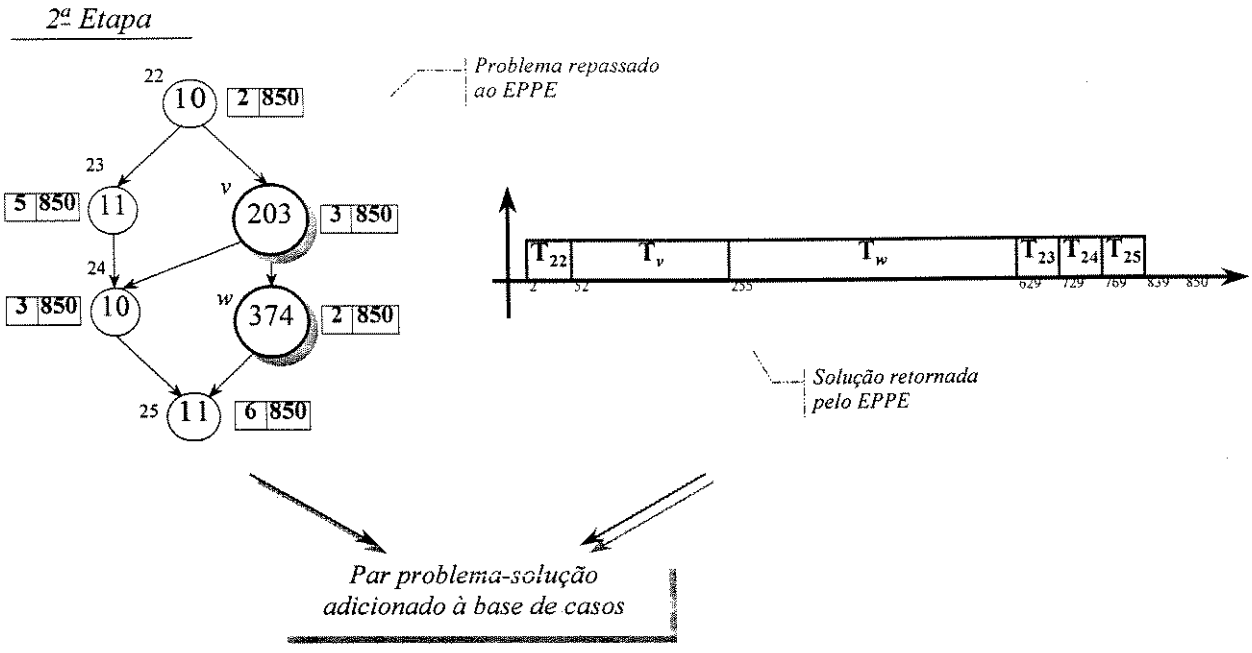



***Figura 47*** *Simplificação do problema P9*

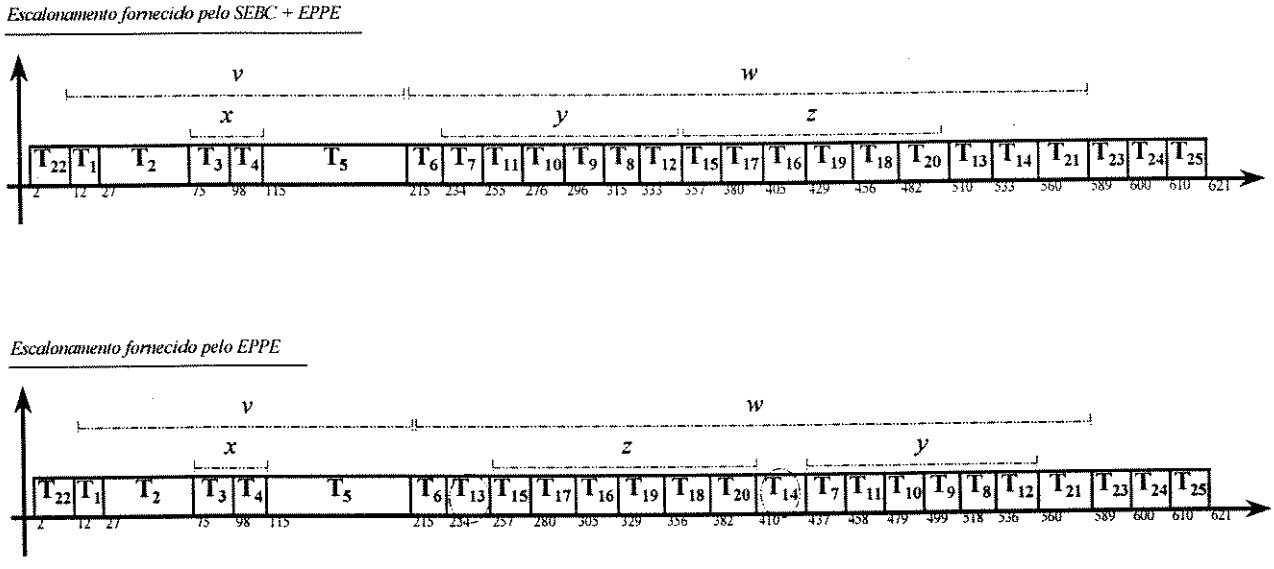



***Figura 48*** *Gráficos de Gantt para a solução final do problema P9*

# 4.4 Desempenho do SEBC

Nesta seção, os resultados obtidos pelo SEBC para as resoluções dos problemas da seção 4.3 são apresentados. Compara-se o desempenho do SEBC com o desempenho do EPPE. O equipamento utilizado para estas medições foi uma *Sun Workstation* com processador *Ultrasparc 1* (167 MHz) e 64 MB de *RAM*.

Os problemas de escalonamento apresentados na seção 4.3 foram resolvidos pelo SEBC, utilizando a base de casos mostrada no Anexo A, e pelo EPPE. A Tabela 5 apresenta os tempos de processamento, em segundos, para as resoluções dos problemas $P_1$ a $P_9$, utilizando o EPPE e o SEBC.

A Figura 49 apresenta uma comparação gráfica dos desempenhos entre o EPPE e o SEBC, utilizando os dados da Tabela 5. Nota-se que o desempenho do SEBC torna-se melhor que o EPPE à medida que cresce o tamanho do problema a ser escalonado e a base de casos possibilita a reutilização de soluções passadas.

O desempenho do SEBC é bastante dependente do tamanho e da qualidade da base de casos, pois quanto mais casos puderem ser recuperados por serem similares aos problemas novos maior será a probabilidade de se ter o problema resolvido apenas a partir do conhecimento disponível.

Quando os subproblemas de problemas submetidos ao SEBC não encontram similares na base, existe a necessidade de se passar o problema simplificado até então para o EPPE tentar resolvê-lo. Feito isto, o problema simplificado em questão é armazenado na base de casos, juntamente com sua solução. Mesmo quando isso é feito (como no exemplo $P_9$), o desempenho do SEBC (utilizando o EPPE) pode ser bem melhor que o desempenho do EPPE sozinho.

***Tabela 5*** *Resultados dos problemas para análise de desempenho entre o SEBC e o EPPE*

| *Problema* | *Tempo de processamento (s)* | |
|---|---|---|
| | *EPPE* | *SEBC* |
| **1** | 0.10 | 0.25 |
| **2** | 0.07 | 0.52 |
| **3** | 0.15 | 0.29 |
| **4** | 0.16 | 0.40 |
| **5** | 0.29 | 0.93 |
| **6** | 0.62 | 0.59 |
| **7** | 1.77 | 1.03 |
| **8** | 6.99 | 4.60 |
| **9** | 4.66 | 2.49 |

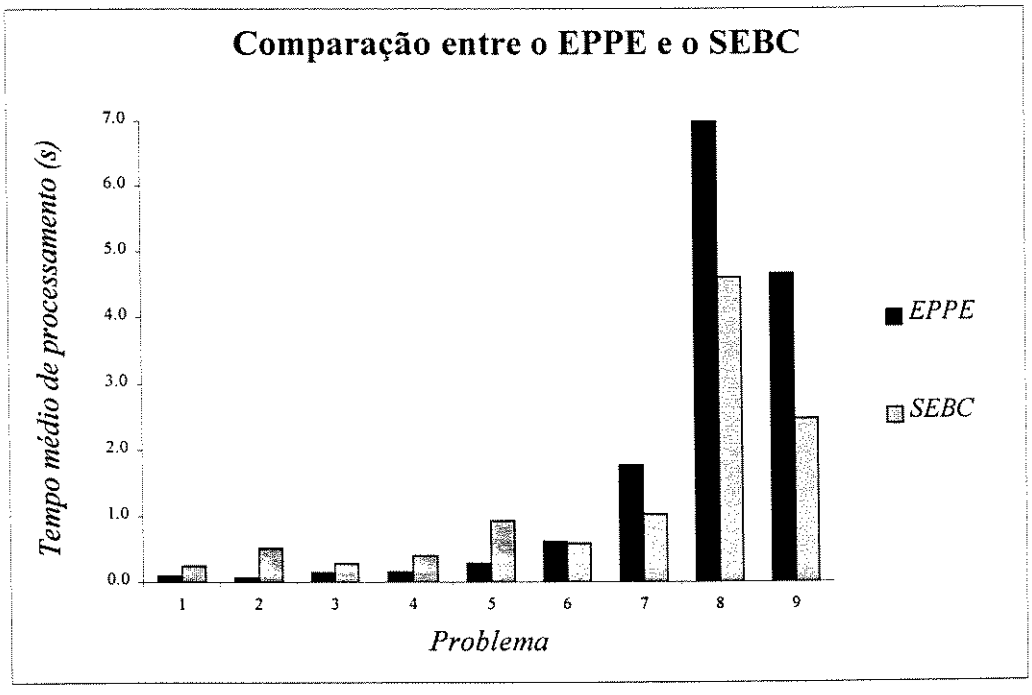



***Figura 49*** *Comparação dos resultados de desempenho entre o SEBC e o EPPE*

A Tabela 6 apresenta os tempos de processamento para as fases de recuperação de casos similares estruturalmente, de simplificação do problema, eventual utilização do algoritmo dedicado (neste trabalho, o EPPE) e o tempo total para a resolução do problema através do SEBC.

Na Figura 50, é apresentado um gráfico de comparação por fases para o SEBC, que mostra, em termos percentuais, as fases de simplificação, recuperação e eventual utilização do EPPE em relação ao tempo total utilizado para cada um dos problemas $P_1$ a $P_9$.

Percebe-se, pela Figura 50, que a fase de recuperação é a que mais sobrecarrega o processamento do SEBC. A eficiência do SEBC é dependente do algoritmo utilizado para a recuperação. O SEBC utiliza o algoritmo de detecção de isomorfismo de subgrafo [Ullman, 1976], implementado por Messmer (1995). À medida que os problemas ficam maiores, os tempos de recuperação na base de casos, juntamente com o tempo de simplificação do problema, começam a ficar menores que os tempos de escalonamento do EPPE. Os problemas $P_1$ a $P_5$ são pequenos, possibilitado o EPPE resolvê-los em menor tempo. Os problemas $P_7$ e $P_8$ são maiores e ilustram este comportamento.

Sabe-se que o problema de detecção de isomorfismo de subgrafo é *NP-completo* [Garey e Johnson, 1979]. Pesquisas nos últimos vinte anos têm mostrado que existem métodos para casamento de grafos que se comportam razoavelmente bem na média em termos de desempenho e tornam-se computacionalmente intratáveis somente em poucas situações. O desempenho prático do algoritmo de Ullman (1976) é bom [Messmer, 1995]. Os resultados do SEBC poderiam ser ainda melhores utilizando-se algoritmos de casamento de grafos mais eficientes, como por exemplo o algoritmo de rede proposto por Messmer (1995), que apresenta ganhos sobre o de Ullman (1976). Contudo, para grafos com menos de quinze nós o desempenho do algoritmo de Ullman (1976) é melhor que o algoritmo de rede. Porém, para grafos maiores o tempo requerido pelo algoritmo de rede cresce menos rapidamente, enquanto o de Ullman (1976) aumenta [Messmer, 1995].

***Tabela 6*** *Resultados por fases do SEBC*

| | ***Tempo de processamento (s)*** | | | |
|---|---|---|---|---|
| ***Problema*** | *Recuperação* | *Simplificação* | *EPPE* | *Tempo Total* |
| **1** | 0.24 | 0.01 | 0 | 0.25 |
| **2** | 0.42 | 0.09 | 0 | 0.51 |
| **3** | 0.26 | 0.02 | 0 | 0.28 |
| **4** | 0.36 | 0.03 | 0 | 0.40 |
| **5** | 0.76 | 0.17 | 0 | 0.93 |
| **6** | 0.51 | 0.06 | 0 | 0.58 |
| **7** | 0.91 | 0.11 | 0 | 1.03 |
| **8** | 3.90 | 0.69 | 0 | 4.60 |
| **9** | 0.94 | 0.11 | 1.43 | 2.49 |

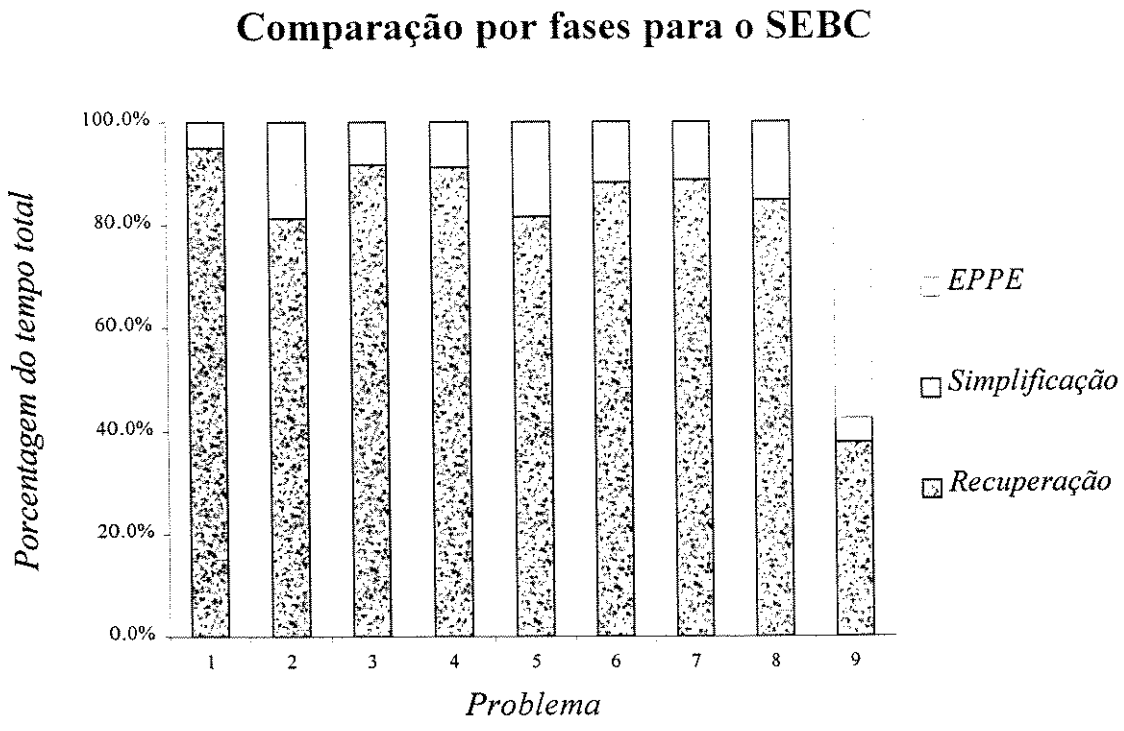



*Figura 50 Comparação por fases para o SEBC*

Através do problema $P_9$, observa-se uma situação de aprendizagem do SEBC. Neste exemplo, o SEBC não encontra uma solução final somente com os casos armazenados na base até o momento, mas simplifica o problema original. Este problema simplificado, que não encontra similar na base, é repassado ao EPPE, o qual retorna uma solução. O SEBC armazena este novo caso (par problema simplificado-solução) na base de casos.

A Tabela 6 mostra os tempos de processamento para o problema $P_9$ que foi submetido ao EPPE e ao SEBC. O SEBC não encontrou uma solução final apenas com os casos que possuía até então na base, mas depois do processo de simplificação entregou ao EPPE um problema mais simples e por isso passível de ser resolvido mais rapidamente. Através da Tabela 5 e da Figura 49 pode-se observar que o SEBC teve um desempenho melhor que o EPPE, mesmo tendo que utilizar o EPPE para solucionar o problema simplificado, que ainda não possuía similar na base de casos.

O procedimento de aprendizagem fornece ao SEBC, ao longo do tempo, o potencial de resolver mais problemas sem a necessidade de utilizar o EPPE, à medida que problemas de uma certa classe podem se tornar mais comuns. Com isto, o tempo de processamento também tende a ser menor. Por exemplo, considerando este novo caso (obtido através da resolução do problema $P_9$) armazenado na base, se surgir uma situação na qual ocorra um problema similar ao problema $P_9$, o SEBC será capaz de resolvê-lo de forma direta com o conhecimento disponível até o momento.

## 4.5 Conclusões

Neste capítulo, foram propostos nove problemas de escalonamento para serem resolvidos pelo SEBC (Sistema de Escalonamento Baseado em Casos, proposto e implementado no capítulo 3) com o intuito de ilustrar a sua utilização e seu potencial. O desempenho do SEBC para os problemas apresentados é comparado com aquele resultante do uso do Escalonador de Processos Periódicos e Esporádicos (EPPE de Melo Jr. (1993)).

Com o intuito de ilustração (análise preliminar), foram considerados problemas (armazenados na base de casos e novos, apresentados ao SEBC) formados a partir das estrutruras básicas apresentadas na Figura 19. Conforme introduzido na seção 3.1, os subproblemas de um novo problema a ser resolvido devem formar um grupo. Respeitando-se esta condição, o SEBC trabalha com problemas e subproblemas com estruturas diversificadas, isto é, relações de precedência e número de tarefas variados. As estruturas variam de duas a seis tarefas e contemplam uma grande variedade de relações de precedência para cada número de tarefas.

Foram criados problemas de escalonamento, a partir das estruturas da Figura 19, com características de tempo de pronto, tempo de execução e prazo de término com grande potencial para constituir subproblemas de novos problemas. Eles foram submetidos ao EPPE, o qual encontrou as respectivas soluções, todas praticáveis. Esses resultados formam os pares problema-solução que constituem a base de casos inicial, descrita no Anexo A.

Os nove problemas propostos encontram-se representados através de grafos acíclicos dirigidos. Os tempo de execução e de pronto e os prazos de término de cada processo, além de suas relações de precedência foram gerados de maneira que os problemas fossem escalonáveis com o objetivo de analisar o desempenho do SEBC relativamente ao EPPE em condições de trabalho efetivo do EPPE (isto é, em situações onde o EPPE necessitasse realizar uma quantidade expressiva de processamento). Uma extensão possível é considerar o SEBC tratando problemas não escalonáveis. Quando o SEBC encontrasse um subproblema do problema novo que fosse similar a um problema não escalonável da base de casos, poderia inferir que o problema apresentado não é escalonável, encerrando o processamento.

Os problemas apresentados neste capítulo foram resolvidos pelo EPPE e pelo SEBC. Para a resolução através do SEBC foram ilustrados detalhes de como o sistema procede, isto é, foram mostradas as etapas de simplificação dos problemas e as adaptações das soluções recuperadas da base de casos. Finalmente, os resultados finais de escalonamento obtidos tanto pelo EPPE como pelo SEBC são apresentados também.

Para a maioria dos problemas, são apresentados dois escalonamentos possíveis, um fornecido pelo SEBC, a partir da simplificação, e o outro fornecido pelo EPPE. Existem algumas diferenças entre os dois resultados. Isto ocorre em função de a política utilizada pelo EPPE escalonar primeiramente a tarefa com maior tempo de execução dentro de um mesmo nível.

O desempenho do SEBC é bastante dependente do tamanho e da qualidade da base de casos, pois quanto mais casos puderem ser recuperados por serem similares aos problemas novos maior será a probabilidade de se ter o problema resolvido apenas a partir do conhecimento disponível.

Quando os subproblemas de problemas submetidos ao SEBC não têm similares na base, existe a necessidade de se passar o problema simplificado até então para o EPPE tentar resolvê-lo. Feito isto, o problema simplificado em questão é armazenado na base de casos, juntamente com sua solução. Mesmo quando isso é feito (como no exemplo $P_9$), o desempenho do SEBC (utilizando o EPPE) pode ser bem melhor que o desempenho do EPPE sozinho.

Nos problemas estudados, à medida que os problemas ficam maiores, os tempos de recuperação na base de casos, juntamente com o tempo de simplificação do problema, começam a ficar menores que os tempos de escalonamento do EPPE, o que sugere a superioridade em termos de desempenho do SEBC nessas situações. Apesar disso, a realização de uma avaliação mais ampla e sistemática é necessária para mais claramente identificar aspectos positivos do SEBC, bem como aspectos que requerem uma maior atenção (deficiências).

# 5

# Conclusões e Perspectivas

O problema de escalonamento representa um domínio interessante a ser abordado, pois é de muita utilidade em várias áreas. A necessidade de se ter tarefas processadas em uma planta de manufatura, clientes de banco aguardando para serem atendidos pelos caixas, aeronaves esperando autorização para pousar e unidades de programa para serem executadas em um computador paralelo ou distribuído têm em comum o problema de escalonamento que surge quando há necessidade de se escolher a ordem na qual as tarefas possam ser executadas e a atribuição de tarefas aos servidores para processamento.

Em Sistemas de Tempo Real Crítico (STRC) com alguma complexidade, os processos a serem escalonados estão sujeitos a um grande número de restrições: tempo de pronto (instante mais cedo no qual o processo torna-se disponível para execução), tempo de execução, prazo de término (prazo máximo para finalização da execução de um dado processo) e relações de precedência.

Uma idéia interessante é armazenar, em uma base de casos, vários problemas já resolvidos e resgatar um ou mais deles, que sejam similares, com suas soluções para auxiliar na resolução de um novo problema. A simplificação deste problema, a partir da reutilização dos casos que são subpartes similares complementam esta idéia. Com este método, torna-se possível reduzir o tempo gasto para se encontrar uma solução. Adicionalmente, quando não existirem casos similares que levem a uma solução direta, pode-se passar um problema de menor complexidade a um algoritmo dedicado, favorecendo a resolução do novo problema proposto. Conseqüentemente, um novo par problema-solução pode incrementar a base de casos existente.

O problema de escalonamento é representado neste trabalho através de grafos acíclicos dirigidos com atributos. Os vértices do grafo representam os processos, seus rótulos indicam quais seus tempos de execução e seus atributos trazem os tempos de pronto e prazo de término. Os arcos determinam as relações de precedência. Neste trabalho, foi proposto e implementado um Sistema de Escalonamento Baseado em Casos (SEBC), que possui como característica principal a idéia de simplificação de um problema através de substituições de seus subproblemas similares a problemas armazenados em uma base de casos e adaptação das soluções destes problemas na busca de uma solução para o problema apresentado.

Um subproblema é uma parte de um problema maior que pode ser resolvida separadamente sem prejuízo para solução do problema como um todo. Os problemas armazenados na base de casos e os problemas apresentados ao SEBC são representados estruturalmente por meio de grafos acíclicos dirigidos com atributos.

Para se utilizar um problema resolvido anteriormente, armazenado na base de casos, é necessário verificar se ele é similar ao novo problema apresentado ao SEBC. Como estão sendo utilizados grafos para a representação dos problemas, é avaliada a similaridade de grafos, ou seja, a similaridade estrutural. O algoritmo de detecção de isomorfismo de subgrafo proposto por Ullman (1976) e implementado por Messmer (1995) através da GUB (*Graph University of Bern*, uma implementação para casamento de grafo) fornece o mecanismo de recuperação de problemas similares estruturalmente aos subproblemas do problema novo.

O SEBC possui como característica principal a idéia de simplificação de um problema através de substituições de seus subproblemas similares a problemas armazenados em uma base de casos e adaptação das soluções destes problemas na busca de uma solução para o problema apresentado. Para que o subproblema seja efetivamente substituído (problema substituível) por um novo processo, devem ser respeitadas relações de tempo de pronto e prazo de término entre os processos do subproblema e do problema recuperado da base para que a solução recuperada seja reutilizável (possa ser adaptada) na solução deste subproblema. Nesta implementação, nota-se que o cálculo do tempo de execução do novo processo criado é "pessimista", isto é, ele pode incluir "vazios" do escalonamento existente para as tarefas do problema armazenado na base de casos. Embora isto possa inviabilizar o escalonamento do novo problema, quando isso não ocorrer, os eventuais "vazios" deixados podem ser usados para escalonar processos aperiódicos.

Quando o SEBC não encontrar uma solução para o problema apresentado somente com os casos disponíveis na base até o momento de determinada resolução, o algoritmo dedicado (EPPE, Escalonador de Processos Periódicos e Esporádicos de Melo Jr. (1993)) é requisitado. O problema simplificado até o momento é repassado ao EPPE que fornece uma solução. Este problema simplificado e sua solução são em seguida armazenados na base de casos, contemplando o processo de aprendizagem de um novo problema do SEBC.

Com o intuito de ilustração (análise preliminar), foram considerados problemas (armazenados na base de casos e novos, apresentados ao SEBC) formados a partir de algumas estrutruras básicas. Os subproblemas de um novo problema a ser resolvido devem formar um grupo. Respeitando-se esta condição, o SEBC trabalha com problemas e subproblemas, gerados automaticamente ou não, com estruturas diversificadas, isto é, relações de precedência e número de tarefas variados. As estruturas variam de duas a seis tarefas e contemplam uma grande variedade de relações de precedência para cada número de tarefas.

Os problemas propostos para a análise preliminar do SEBC encontram-se representados através de grafos acíclicos dirigidos. Os tempos de execução e de pronto e os prazos de término de cada processo, além de suas relações de precedência foram gerados de maneira que os problemas fossem escalonáveis com o objetivo de analisar o desempenho do SEBC relativamente ao EPPE. Os processos dos problemas já estão segmentados, por isso os prazos de término (*deadlines*) são idênticos.

O desempenho do SEBC é bastante dependente do tamanho e da qualidade da base de casos, pois quanto mais casos puderem ser recuperados por serem similares aos problemas

novos maior será a probabilidade de se ter o problema resolvido apenas a partir do conhecimento disponível.

Quando os subproblemas de problemas submetidos ao SEBC não encontram similares na base, existe a necessidade de se passar o problema simplificado até então para o EPPE tentar resolvê-lo. Feito isto, o problema simplificado em questão é armazenado na base de casos, juntamente com sua solução. Mesmo quando isso é feito, o desempenho do SEBC (utilizando o EPPE) pode ser bem melhor que o desempenho do EPPE sozinho, como mostrado através de um exemplo. Considerando a extensão para problemas não escalonáveis, o processo de armazenamento de um novo caso na base poderia ocorrer, armazenando-se o problema não escalonável em questão.

À medida que os problemas ficam mais complexos quanto às relações de precedência, por exemplo, experimentos realizados mostram que os tempos de recuperação na base de casos, juntamente com o tempo de simplificação do problema, tendem a ficar menores que os tempos de escalonamento do EPPE, o que sugere a superioridade em termos de desempenho do SEBC nessas situações. Apesar disso, a realização de uma avaliação mais ampla e sistemática é necessária para mais claramente identificar aspectos positivos do SEBC, bem como aspectos que requerem uma maior atenção (deficiências).

A abordagem baseada em casos contendo escalonamentos, empregada neste trabalho, se assemelha com aquela proposta por Cunningham e Smyth (1996), que propuseram a reutilização de componentes armazenados em uma base. A idéia de simplificação do SEBC a partir de subproblemas com similares em uma base se parece com a de Cunningham e Smyth (1996), mas foi concebida independentemente dele, isto é, antes de se conhecer os resultados de sua pesquisa.

Apesar das semelhanças existentes, aspectos importantes diferenciam este trabalho do de Cunningham e Smyth (1996). Entre eles, podemos destacar o tipo de problema de escalonamento tratado e a abordagem e técnicas usadas para construir a base de casos e recuperar casos passados.

O problema de escalonamento abordado por Cunningham e Smyth (1996) envolve o escalonamento de $N$ tarefas em uma máquina única em uma situação onde o tempo de pronto da tarefa é dependente da seqüência. Este problema se refere à montagem de produtos eletrônicos onde o tempo de pronto da tarefa em máquinas de seqüenciamento de componentes usadas em montagem de placas de circuito impresso é dependente de escalonamento. O tempo de pronto associado com o escalonamento da tarefa B depois da tarefa A depende diretamente do número de componentes requeridos em B que não foram usados em A [Cunningham e Browne, 1986].

O problema abordado neste trabalho é o escalonamento de processos periódicos em um STRC monoprocessado. Em particular, o problema considerado aqui possui as seguintes características: restrições de tempo, periodicidade e relações de precedência.

Cunningham e Smyth (1996) utilizam duas abordagens diferentes para a recuperação: a primeira consiste em selecionar um único caso o qual é reusado como um esquema para a solução objetivada; a segunda, retorna uma coleção de seqüências de solução, sendo que cada uma delas pode ser usada sem alterações para produzir uma parte do escalonamento objetivado.

A abordagem para recuperação utilizada neste trabalho retorna um conjunto de problema-solução, cujos problemas são similares estruturalmente a subproblemas do problema novo. A solução recuperada é em seguida adaptada e reutilizada na substituição do subproblema por um novo processo (esta situação ocorre no procedimento de simplificação do problema novo apresentado ao SEBC).

Certamente, o sistema proposto e implementado neste trabalho pode ser avaliado através da observação de critérios diversificados. Ele pode também ser estendido com o intuito de melhorar sua eficiência. Portanto, existe a possibilidade de contribuições futuras que explorem seu potencial. A seguir, são apresentadas algumas sugestões para continuidade deste trabalho:

- ✧ análise experimental do SEBC, incluindo:

  1. geração de casos de teste de forma sistemática, variando diversos parâmetros, tais como folga das tarefas, etc; e

  2. criação/manutenção da base de casos de forma automática/sistemática.

- ✧ utilização de novos algoritmos de casamento de grafo para recuperação de casos

- ✧ propiciar substituições de subproblemas que não sejam pessimistas

- ✧ extensão para tratamento de problemas não escalonáveis

- ✧ tratamento da relação de exclusão mútua entre os processos

- ✧ utilização de diferentes algoritmos dedicados, dependendo das variações de parâmetros do problema

- ✧ classificação dos problemas na entrada do sistema para otimizar a recuperação de casos

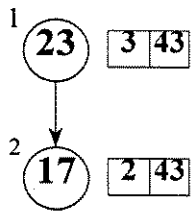
# Anexo A

# Base Inicial para Análise do SEBC

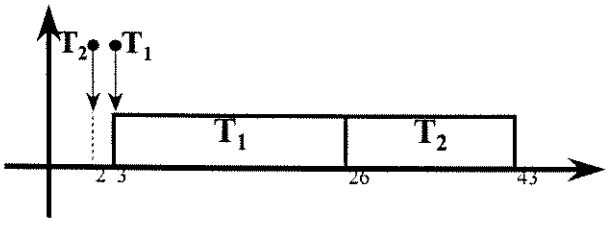
Casos (pares problema-solução) da base de casos inicial, utilizados na resolução dos problemas propostos no capítulo 4. Os tipos de estruturas básicas são descritos na seção 4.1.

***Casos Utilizando o Tipo de Estrutura 1***

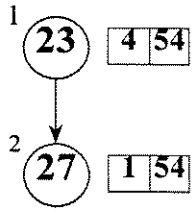
*Problema 1*



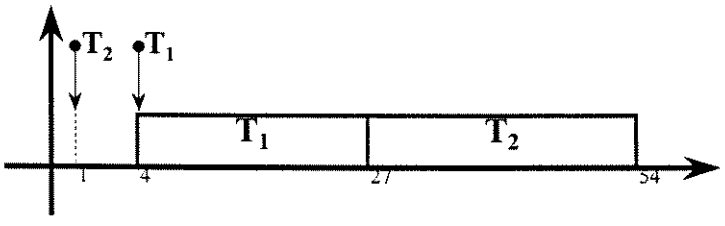

*Solução 1*



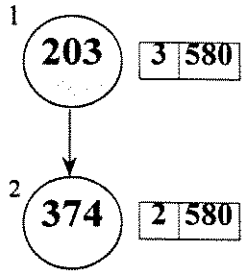

*Problema 2*




*Solução 2*




*Problema 3*




*Solução 3*

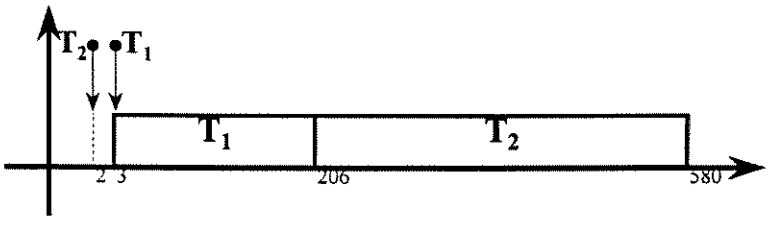
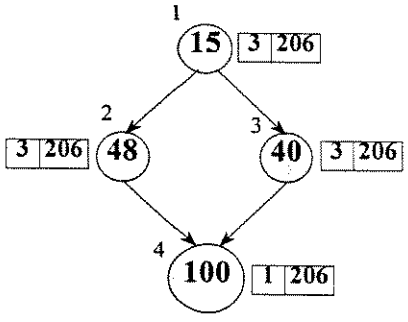


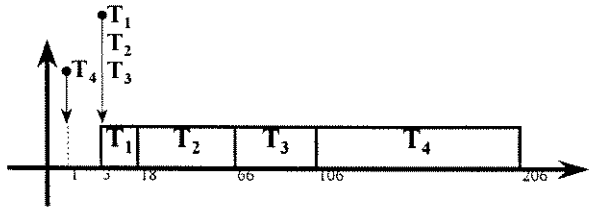

***Figura A.1*** *Casos da base inicial com tipo de estrutura 1*

## *Casos Utilizando o Tipo de Estrutura 2*

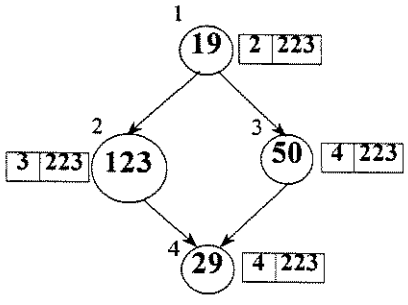
*Problema 1*



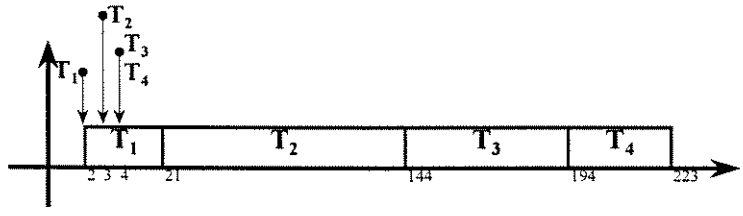

*Solução 1*



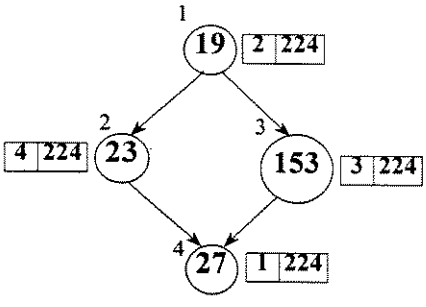

*Problema 2*



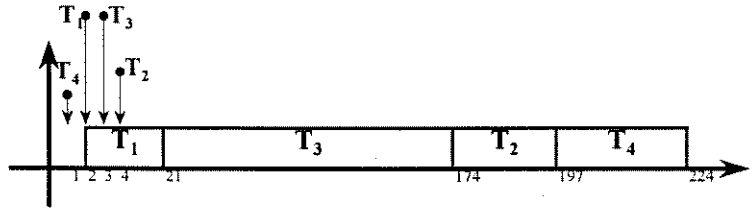

*Solução 2*



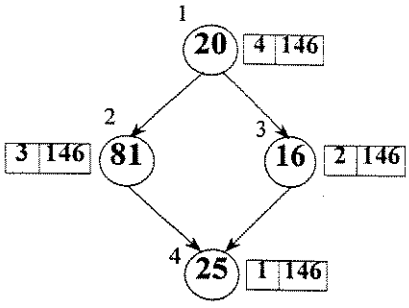

*Problema 3*



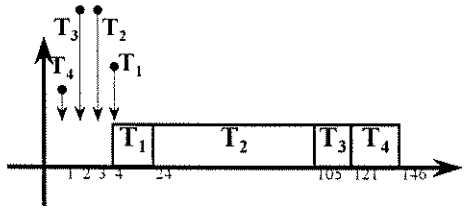

*Solução 3*



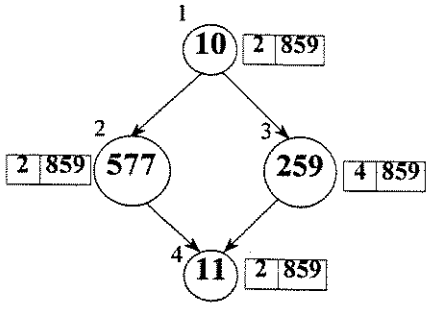
*Problema 4*




*Solução 4*




*Problema 5*




*Solução 5*

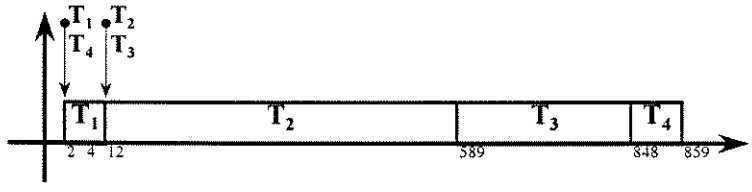
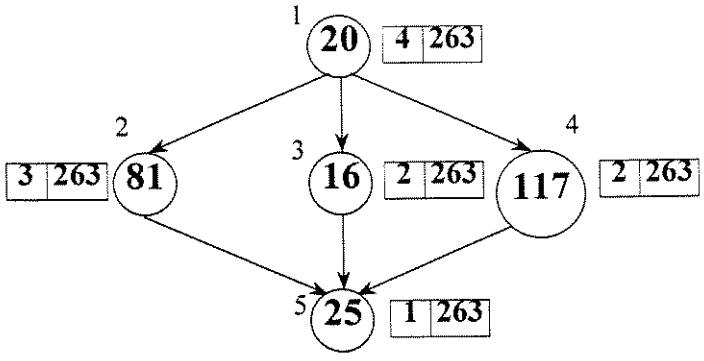



***Figura A.2*** *Casos da base inicial com tipo de estrutura 2*

### *Caso Utilizando o Tipo de Estrutura 3*

*Problema 1*




*Solução 1*

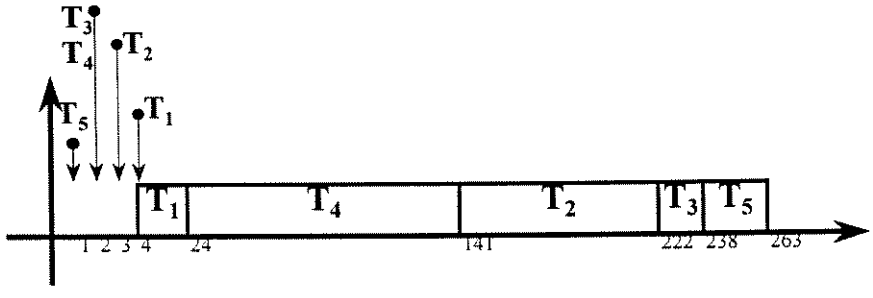
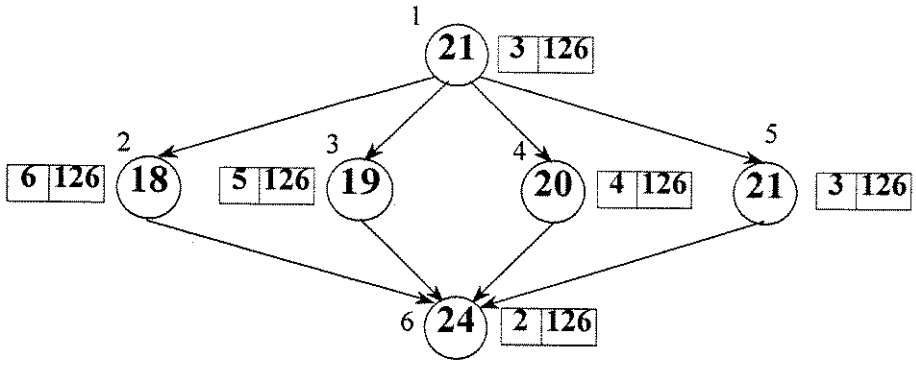



***Figura A.3*** *Caso da base inicial com tipo de estrutura 3*

### *Caso Utilizando o Tipo de Estrutura 4*

*Problema 1*




*Solução 1*

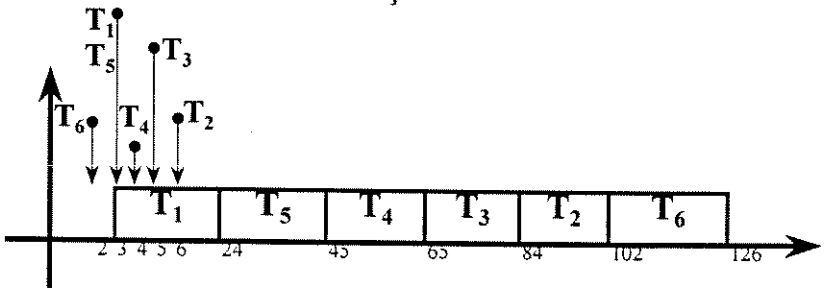
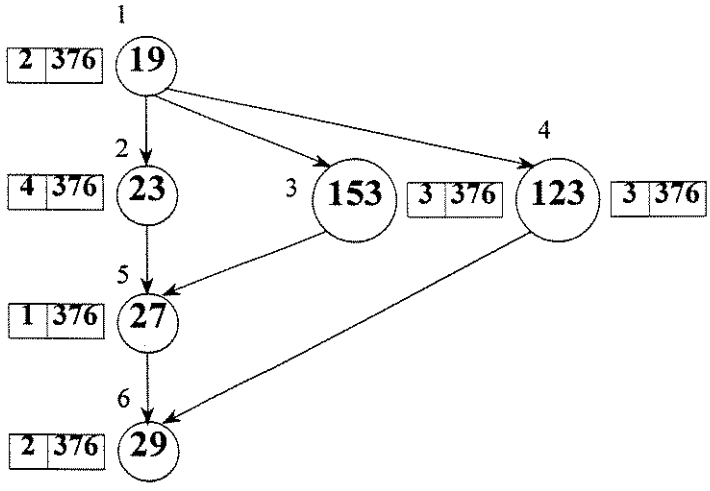



***Figura A.4*** *Caso da base inicial com tipo de estrutura 4*

### *Caso Utilizando o Tipo de Estrutura 5*

*Problema 1*




*Solução 1*

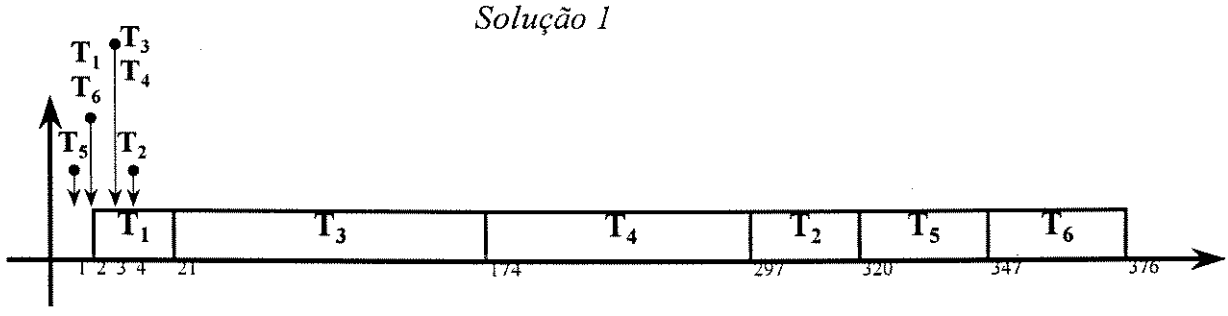
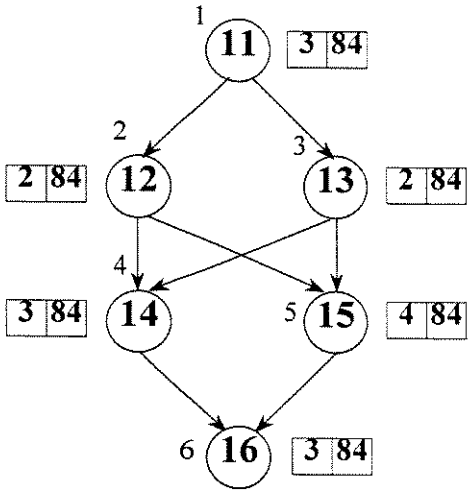



***Figura A.5*** *Caso da base inicial com tipo de estrutura 5*

### *Caso Utilizando o Tipo de Estrutura 6*

*Problema 1*




*Solução 1*

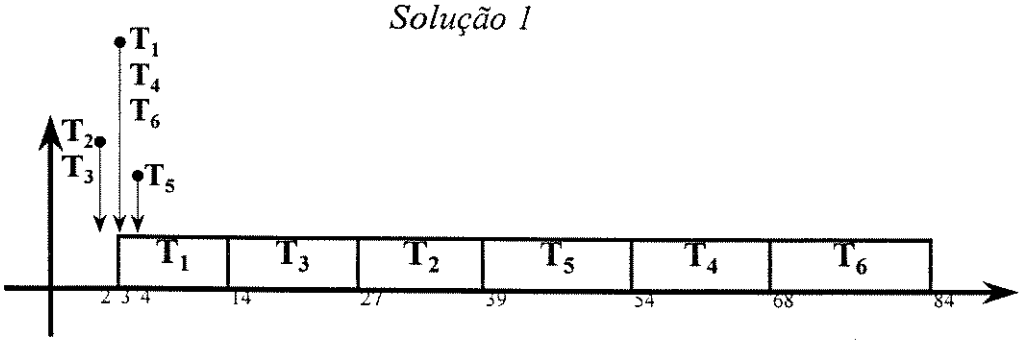
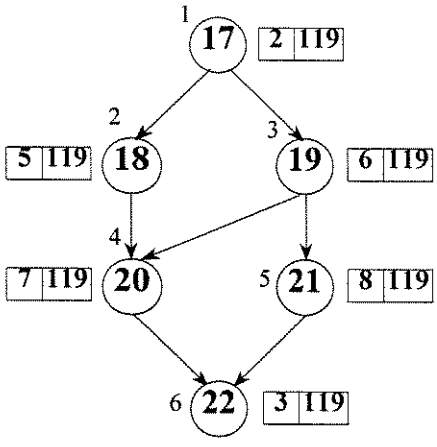



***Figura A.6*** *Caso da base inicial com tipo de estrutura 6*

### *Caso Utilizando o Tipo de Estrutura 7*

*Problema 1*




*Solução 1*

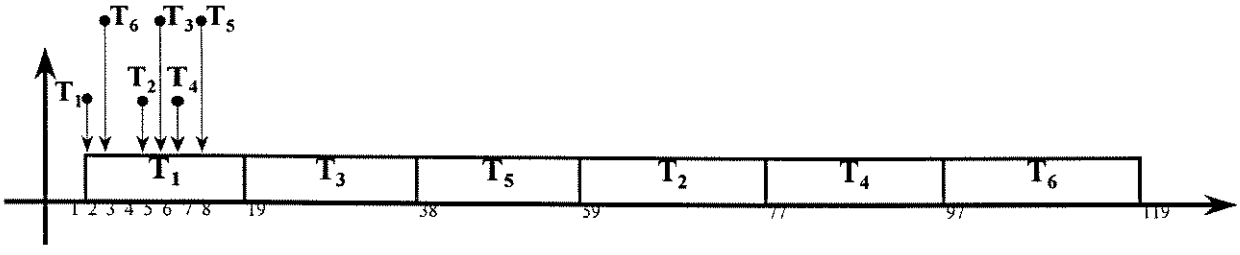
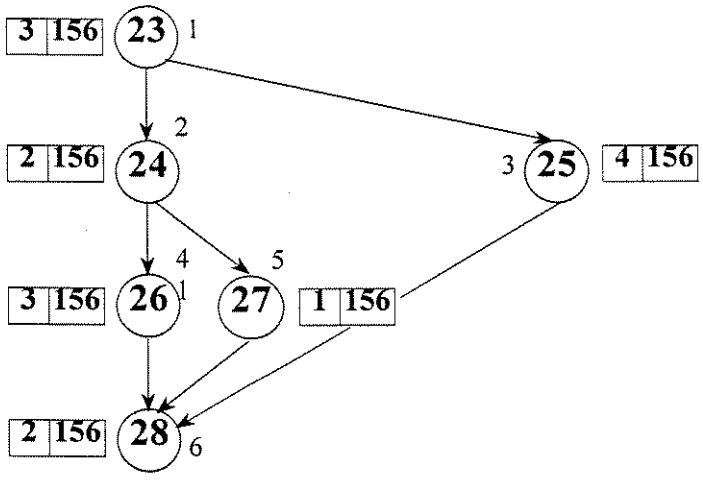



***Figura A.7*** *Caso da base inicial com tipo de estrutura 7*

### *Caso Utilizando o Tipo de Estrutura 8*

*Problema 1*




*Solução 1*

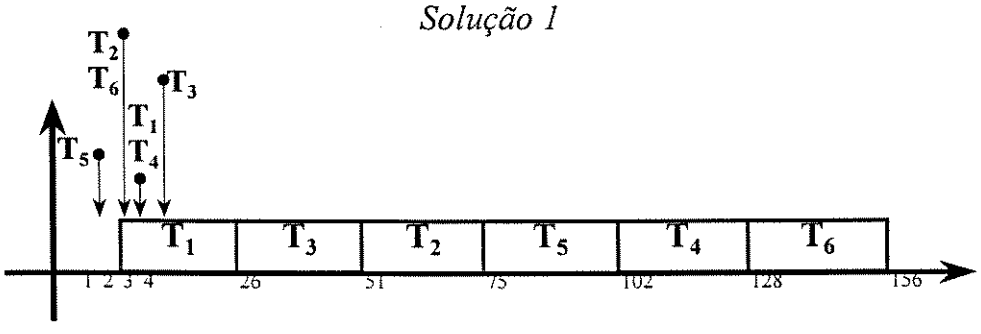



***Figura A.8*** *Caso da base inicial com tipo de estrutura 8*

# Referências Bibliográficas


Aamodt, A. e Plaza, E. 1996. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *AICom-Artificial Intelligence Communications*, v. 7, n. 1, pp. 1-32.

Alterman, R. 1986. An adaptive planner. In *Proceedings of AAAI-86*, pp. 65-69, AAAI, Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Ashley, K.D. 1987. Distinguishing – a reasoner's wedge. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 737-747, Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, N.J.

Bain, W.M. 1986. Case-Based Reasoning: A Computer Model of Subjective Assessment. Phd thesis, Yale University.

Bareiss, E.R. 1989. Exemplar-based knowledge acquisition: A unified approach to concept representation, classification, and learning. Boston: Academic Press.

Barletta, R. e Hennessy, D. 1989. Case adaptation in autoclave layout design. In *Proceedings: Workshop on case-based reasoning (DARPA)*, Pensacola Beach, Florida. San Mateo, CA: Morgan Kaufmann.

Ben-Ari, M. 1990. Principles of Concurrent and Distributed Programming, UK: Prentice Hall International Ltd. .

Cheng, S.C. e Stankovic, J.A. 1987. Scheduling Algorithms for Hard Real-Time Systems – A Brief Survey, *Real-time Systems Newsletter*, v. 3, n. 2, pp. 1-24.

Collins, G.C. 1987. Plan Creation: Using Strategies as Blueprints. PhD thesis, Yale University.

Cunningham, P. e Browne, J. 1986. A LISP-based heuristic scheduler for automatic insertion in electronics assembly, *International Journal of Production Research*, 24, pp. 1395-1408.

Cunningham, P. e Smyth, B. 1996. CBR in scheduling: reusing solution components. Technical Report TCD-CS-96-12, Department of Computer Science, Trinity College Dublin.

El-Rewini, H. e Ali, H.H. 1995. Task Scheduling in Multiprocessing Systems, *IEEE Computer*, December, pp. 27-37.

French, S. 1986. Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop, New York, Ellis Horwood Limited.

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

Gentner, D. 1989. Finding the needle: Accessing and reasoning from prior cases. In *Proceedings: Workshop on case-based reasoning (DARPA)*, Pensacola Beach, Florida. San Mateo, CA: Morgan Kaufmann.

Gick, M. e Holyoak, K. 1980. Analogical problem solving. *Cognitive Psychology* 12: pp. 306-355.

Goodman, M. 1989. CBR in battle planning. In *Proceedings: Workshop on case-based reasoning (DARPA)*, Pensacola Beach, Florida. San Mateo, CA: Morgan Kaufmann.

Hammond, K.J. 1989. Case-based Planning: Viewing Planning as a Memory Task. Perspectives in Artificial Intelligence, Academic Press, Boston, MA.

Holyoak, K.J. 1985. The pragmatics of analogical transfer. *The psychology of learning and motivation*, ed. G. Bower. New York: Academic Press.

Klein, G.A. e Calderwood, R. 1988. How do people use analogues to make decisions? In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*. Northvale, NJ: Erlbaum.

Kolodner, J.L. 1984. Retrieval and Organizational Strategies in Conceptual Memory. Lawrence Erlbaum Associates, Hillsdale, N.J.

Kolodner, J.L. 1991. Improving Human Decision Making through Case-Based Decision Aiding, *AI Magazine*, Summer, pp. 52-68.

Kolodner, J.L. 1993. *Case-Based Reasoning*. Georgia Institute of Technology. Morgan Kaufmann Publishers.

Kopeikina, L. et al. 1988. Case-Based reasoning for continuous control. In *Proceedings: Workshop on case-based reasoning (DARPA)*, Clearwater, Florida. San Mateo, CA: Morgan Kaufmann.

Koton, P. 1988. Reasoning about evidence in causal explanation. In *Proceedings of AAAI-88*, pp. 256-261. Cambridge, MA: AAAI Press/MIT Press.

Koton, P. 1989a. Using experience in learning and problem solving. PhD thesis, Department of Computer Science, MIT.

Koton, P. 1989b. SMARTplan: A Case-Based Resource Allocation and Scheduling System. In *Proceedings of the Case-Based Reasoning Workshop*, 285-289, Morgan Kaufmann, Publishers.

Lancaster, J.S. e Kolodner, J. 1987. Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. Northvale, NJ: Erlbaum.

Lancaster, J.S. e Kolodner, J.L. 1988. Varieties of learning from problem solving experience. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*. Northvale, NJ: Erlbaum.

Lebowitz, M. 1980. Generalization and Memory in an Integrated Understanding System. Research Report 186, Yale University.

Leung, J.Y.T e Merril, M.L. 1980. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks, *Information Processing Letters*, v. 11, n. 3, pp. 115-118.

Maurer, F. 1993. Similarity based retrieval of interpretation models. In *M.M. Richter, S. Wess, K.D. Althoff and F. Maurer, editors, Preproceedings: First European Workshop on Case-Based Reasoning*, pp. 366-370.

Melo Jr., A. 1993. Uma estratégia de escalonamento de processos periódicos e esporádicos em sistemas de tempo real crítico monoprocessados. Dissertação de Mestrado, Campinas, Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

Messmer, B.T. 1995. Efficient graph matching algorithms. PhD Thesis, Switzerland, Institute of Computer Science and Applied Mathematics, University of Bern.

Miyashita, K. e Sycara, K. 1995. CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair. Technical Report CMU-RI-TR-95-34, The Robotics Institute, School of Computer Science, Carnegie-Mellon University.

Mok, A.K. 1983. Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment, PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge.

Myaeng, S.H. e Lopez-Lopez, A. 1992. Conceptual graph matching: a flexible algorithm and experiments. *Journal of Experimental and Theoretical Artificial Intelligence*, v. 4, n. 4, pp.107-126.

Porter, B. et al. 1990. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence* 45: pp. 229-263.

Read, S. e Cesa, I. 1990. This reminds me of the time when ...: Expectation failures in reminding and explanation. *Journal of Experimental Social Psychology*, 26.

Riesbeck, C.K. e Schank, R.C. 1989. *Inside Case-Based Reasoning*. The Institute for the Learning Sciences, Northwestern University. Evanston, Illinois : Lawrence Erlbaum Associates, Publishers.

Rissland, E.L. e Ashley, K.D. 1986. Hypotheticals as heuristic device. In *Proceedings of AAAI-86*, pp. 289-297, AAAI, Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Ross, B.H. 1986. Remindings in learning: Objects and tools. In *Similarity, analogy, and thought*, ed. S. Vosniadou and A. Ortony. New York: Cambridge University Press.

Ross, B.H. 1989. Some psychological results on case-based reasoning. In *Proceedings: Workshop on case-based reasoning (DARPA)*, Pensacola Beach, Florida. San Mateo, CA: Morgan Kaufmann.

Schank, R.C. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press.

Simoudis, E. 1992. Using case-based retrieval for customer techinical support. *IEEE Expert*, v.7, n. 5, pp. 7-13.

Simpson, R.L. 1985. A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation. Technical Report GIT-ICS-85/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta GA.

Sha, L. e Goodenough, J.B. 1990. Real-Time Scheduling – Theory and Ada, *Computer*, April, pp. 53-62.

Sprunt, B. et al 1989. Aperiodic Task Scheduling for Hard Real-Time Systems, *The Journal of Real-Time Systems*, v. 1, n. 1, pp. 27-60.

Stankovic, J.A. et al 1985. Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems, *IEEE Transactions on Computers*, v. C-34, n. 12, pp. 1130-1143.

Sycara, E.P. 1987. Resolving Adversarial Conflicts: An Approach to Integrating Case-Based and Analytic Methods. Technical Report GIT-ICS-87/26, Georgia Institute of Technology, School of Information and Computer Science, Atlanta GA.

Sycara, K. e Miyashita, K. 1994. Case-Based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains. In *Proceedings of the 12th National Conference on Artificial Intelligence*, Cambridge, MA, AAAI Press/MIT Press, pp. 44-49.

Ullmann, J. R. 1976. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, v. 23, n. 1, pp. 31- 42.

Xu, J. e Parnas, D.L. 1990. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, v. 16, n. 3, pp. 360-369.

Xu, J. e Parnas, D.L. 1991. On Satisfying Timing Constraints in Hard Real-Time Systems, *Software Engineering Notes*, v. 16, n. 5, pp. 132-146.