Uma Metodologia de Especificação de Times Assíncronos

Hélvio Pereira Peixoto
helvio@dcc.unicamp.br
Pedro Sérgio de Souza
pss@dcc.unicamp.br

Relatório Técnico DCC–94-12

Novembro de 1994

Uma Metodologia de Especificação de Times Assíncronos

Hélvio Pereira Peixoto
helvio@dcc.unicamp.br

Pedro Sérgio de Souza
pss@dcc.unicamp.br

Universidade Estadual de Campinas
Departamento de Ciência da Computação - IMECC
Cx.Postal - 6065 CEP - 13081.970
Campinas - SP

Sumário

Times Assíncronos *(A-Teams)* perfazem uma nova técnica de resolução de problemas baseada na utilização simultânea de diversos algoritmos heurísticos, que cooperam entre si de maneira sinérgica, encontrando soluções ótimas ou quase ótimas, as quais não seriam encontradas pelos algoritmos isoladamente. Por se tratar de técnica recente, as especificações de *A-Teams* até o momento têm ocorrido empiricamente. Este artigo descreve uma metodologia para a especificação de *A-Teams*, facilitando a sua utilização na resolução dos problemas de Otimização Combinatória.

Abstract

Asyncronous Teams (A-Teams) are a new problem resolution technique that uses simultaneously various heuristic algorythms. These algorythms cooperate synergetically one with the other to find optimal or nearly optimal solutions that would not be found through isolated algorythms. Being a new technique, A-Teams especifications have until now happened empirically. This paper describes a methodology to specify A-Teams, turning its usage to resolve Combinatorial Optimization Problems easier.

Palavras-Chave: Métodos Aproximados e Otimização Combinatória.

1 Introdução

Infelizmente, grande parte dos problemas de Otimização Combinatória não possuem algoritmos que os resolvam eficientemente (os problemas NP-Difíceis). Visto que é improvável a existência de tais algoritmos [GJ79], muitas técnicas surgiram com intuito de minorar o esforço computacional necessário para a obtenção de soluções, a princípio, ótimas. Uma técnica recentemente introduzida por Souza [Sou93], os Times Assíncronos (do inglês *Asyncronous Teams* ou *A-Teams*), tem sido aplicada com sucesso em problemas de grande porte [Sou93, Mur92, Che92].

Este artigo tem por finalidade a descrição de uma metodologia de especificação de *A-Teams*, facilitando a sua utilização na resolução dos problemas de Otimização Combinatória.

2 A-Teams

Um *A-Team* é uma maneira recente de organizar diversos softwares (agentes) para resolver um dado problema. Na forma em que foi concebido em 1992 por Talukdar e Souza [TS92], um *A-Team* consiste num time de agentes que trabalham de maneira assíncrona, interativa e cíclica, sobre um conjunto de dados depositados em memórias compartilhadas.

As memórias armazenam soluções ou informações relevantes à resolução do problema, e o fato de serem compartilhadas garante que os resultados produzidos por um agente estarão disponíveis aos demais. Os agentes podem ler e escrever nas memórias compartilhadas sempre que desejarem, conforme suas respectivas políticas de acesso às mesmas. No *A-Team*, cada agente tem a oportunidade de escrever sua melhor solução nas memórias, que pode então ser usada como entrada por outro agente do time. Esta interação entre os agentes leva à necessidade da existência de fluxo cíclico das soluções. Estes ciclos permitirão *feedback* e a possibilidade de um agente operar sobre uma solução criada previamente por ele e modificada pelos demais. Esta cooperação entre os agentes aumenta as chances do time gerar soluções melhores, que não seriam geradas pelos agentes isoladamente.

Um agente consiste em um algoritmo que se propõe a resolver o problema (ou parte dele) e um protocolo de comunicação com a memória. Geralmente, devido ao porte e complexidade dos problemas tratados, os agentes despendem muito mais tempo processando dados que efetuando comunicações com as memórias. Esta característica, aliada à autonomia dos agentes e ao assincronismo das comunicações, torna os *A-Teams* adequados ao processamento paralelo e distribuído.

Nos *A-Teams* não existem agentes surpevisores. Cada agente é livre para escolher quando e que solução processar. Conseqüentemente, agentes podem entrar e sair do time a qualquer instante. A saída de um agente causa apenas uma degradação suave no time, não comprometendo de todo a geração de novas soluções.

Uma vez que as memórias compartilhadas não podem incorporar soluções indefinidamente, a cada uma estão associados um ou mais agentes denominados *Destruidores*. As únicas funções de um Destruidor são julgar e eliminar, baseado em uma política de destruição, que solução eliminar a fim de abrir espaço a uma nova. Esta política de destruição é geralmente relacionada à qualidade dos dados. Nos *A-Teams*, tão importante quanto a tarefa de produzir boas soluções é a de eliminar soluções não promissoras.

A aplicação de *A-Teams* a diversos domínios tem demonstrado que esta técnica pode ser extremamente eficiente na resolução de problemas considerados difíceis. Um exemplo é o problema do Caixeiro Viajante. Este tem sido um dos problemas clássicos da literatura mais estudados, devido a sua simplicidade na formulação, dificuldade em se obter soluções ótimas (é NP-Difícil) e as inúmeras aplicações práticas. O problema consiste em encontrar o menor circuito Hamiltoniano, dado um conjunto de cidades e as distâncias entre as mesmas. Souza construiu um *A-Team* para o problema do Caixeiro Viajante usando, basicamente, heurísticas encontradas na literatura. Este *A-Team* encontrou soluções ótimas para todas as instâncias testadas, enquanto que os mesmos algoritmos, executados isoladamente, só encontraram soluções ótimas para a menor das instâcias (100 cidades).

3 Uma Metodologia de Especificação de A-Teams

Embora a idéia que fundamente os *A-Teams* seja simples, pergunta-se: Quais as características de um problema que indicam a adequabilidade ou não do uso de *A-Teams*? Como conceber e criar um *A-Team*? Haveria estruturas prévias de *A-Teams* para algumas classes de problemas? Quais seriam estas classes e as respectivas estruturas? Como implementar um *A-Team*?

Nossos estudos, até o momento, permitiram a identificação de alguns passos para a resolução de problemas, particularmente através da especificação e construção de *A-Teams*.

Especificação do Problema

A primeira tarefa na resolução de qualquer problema é sua correta identificação e especificação. Todas as variáveis, seus domínios e restrições devem ser estabelecidos nesta fase.

Definição da Representação e Padrão de Qualidade das Soluções

O passo subseqüente à definição do problema consiste na definição dos resultados esperados. Uma vez que os *A-Teams* operam sobre soluções, é preciso definir claramente qual é a estrutura de uma solução válida. Este passo facilitará futuramente a especificação das memórias. Ainda relacionado às soluções, qual o nível de qualidade desejado? Embora estejamos sempre em busca do ótimo, para muitos problemas as soluções esperadas não são necessariamente as soluções ótimas. Este pode ser o caso de problemas de funções multi-objetivas, onde o usuário pode escolher dentre um conjunto de soluções "boas" a que julgar mais adequada. Existem ainda os problemas para os quais não é fácil (ou mesmo possível) modelar quantitativamente a qualidade de uma solução. Um bom exemplo são os problemas de *Design*, onde a estética é um fator importante e subjetivo.

Identificação dos Algoritmos Disponíveis

Uma vez que está definido o problema, a representação de uma solução e o padrão de qualidade associado, o passo seguinte é a coleta dos diversos métodos disponíveis na literatura para resolvê-lo. Ao identificar tais métodos e suas respectivas características, pode-se classificar os problemas em três classes: classe dos problemas que possuem ao menos um Algoritmo Adequado, classe dos problemas que possuem Algoritmos Inadequados e classe dos problemas que possuem apenas Algoritmos Inaptos. Esta classificação não se refere à complexidade dos algoritmos, mas sim a habilidade dos algoritmos em prover as soluções desejadas, respeitando todas as restrições dos problema, inclusive restrições em relação ao tempo disponível para resolvê-los.

Os *Algoritmos Adequados* correspondem aos algoritmos que encontram as soluções desejadas no tempo disponível. Exemplo: algoritmo *QuickSort* de ordenação de vetores, Simplex para os problemas de programação linear e *Branch-and-Bound* para os problemas de programação inteira, considerando que todos eles tenham tempo suficiente para encontrar a solução desejada. Os *Algoritmos Inadequados* são aqueles que, na maioria das vezes, não satisfazem o padrão de qualidade desejado, gerando soluções aproximadas ou com pequenas

violações das restrições do problema. Exemplo: heurísticas (que geram soluções aproximadas) e algoritmos de relaxação (que geram soluções infactíveis que podem ser facilmente ajustadas a soluções factíveis). A classe dos problemas de *Algoritmos Inaptos* corresponde aos problemas para os quais os algoritmos disponíveis são capazes de encontrar apenas soluções profundamente infactíveis. Um exemplo é o problema de equações diferencias não lineares de milhões de variáveis em aplicações de previsão do tempo, onde as variáveis são modificadas rápida e periodicamente.

Muitos problemas de aplicações práticas se enquadram na classe dos problemas de Algoritmos Inadequados. Isto decorre do tempo limitado (geralmente pequeno) de que dispomos para resolvê-los. Este é o domínio de problemas que abordaremos neste trabalho, em particular os problemas de Otimização Combinatória.

Uma Estrutura Inicial

Geralmente, os problemas da classe dos Algoritmos Inadequados possuem várias heurísticas, que se propõem a resolvê-los aproximadamente, uma vez que são improváveis que existam algoritmos exatos polinomiais [GJ79]. Este é um fator coadjuvante na construção dos agentes, pois as heurísticas, quase sempre, são simples e fáceis de serem implementadas. Porém, resta a decisão sobre que heurística usar e como combiná-las. Para combinar heurísticas, é necessário primeiramente classificá-las, o que é feito segundo a abordagem utilizada na obtenção de uma solução:

A: Heurísticas de Construção. A partir de uma solução parcial (incompleta ou nula) do problema, estes algoritmos constróem uma solução válida e completa elemento a elemento.

B: Algoritmos de Consenso. Geram uma solução parcial a partir de uma ou mais soluções válidas e completas. Tentam capturar um "consenso" de qualidade, gerando soluções parciais. O princípio utilizado pelos algoritmos de consenso é o seguinte: se duas soluções consideradas boas possuem fragmentos comums, então estes fragmentos possuem grandes chances de estarem em alguma solução ótima.

C: Heurísticas de Relaxamento. Obtêm soluções para o problema original de alguma forma relaxado. Úteis para gerar limites inferiores e soluções aproximadas.

D: Algoritmos de Factibilidade. Ajustam soluções de problemas relaxados para serem válidas ao problema original.

E: Heurísticas de Melhoria. Partindo de uma solução válida, este tipo de heurística faz modificações nesta solução à procura de soluções melhores.

F: Algoritmos de Particionamento. Divide o problema original em uma seqüência de sub-problemas distintos e mais facilmente solucionáveis do que o problema original.

G: Algoritmos de Composição. Dado que algoritmos de particionamento geraram sub-soluções para diversos sub-problemas, os algoritmos de composição utilizam destas sub-soluções para gerar soluções válidas para o problema original.

I: Iterativo. Dependendo do problema, informações fornecidas por um elemento humano podem ser de grande valia no processo de resolução do problema. Estes algoritmos permitem a introdução de tais informações nas soluções correntes do problema.

J: Exatos. Uma vez que limites inferiores e superiores muito próximos à solução ótima sejam obtidos, torna-se potencialmente viável a utilização de métodos exatos, mesmo para os problemas NP-Difícies.

K: Dual-Primal. Algoritmos que transformam uma solução dual em primal.

L: Primal-Dual. Algoritmos que transformam uma solução primal em dual.

Uma possível estrutura geral de um *A-Team* envolvendo todas as classes de algoritmos descritas acima está representada na figura 1. Nesta figura, os retângulos representam as memórias compartilhadas e as setas, as classes de algoritmos mencionadas. Em cada classe podem existir vários algoritmos que lêem soluções da memória de onde a seta sai e escrevem na memória onde a seta chega.

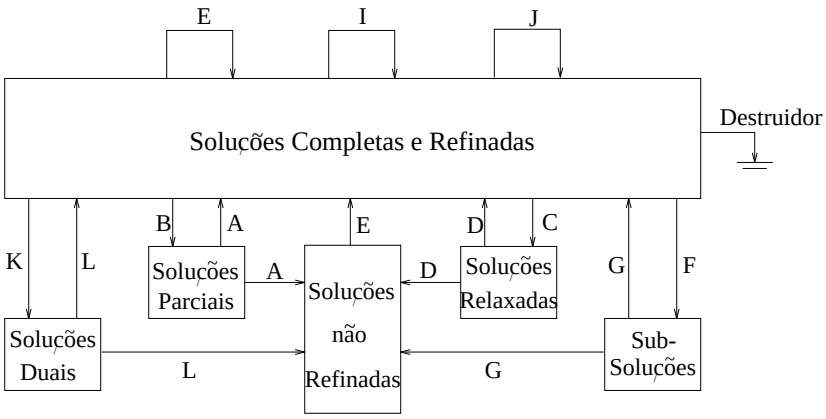

Figura 1: Uma estrutura geral para *A-Teams*

A memória de Soluções Completas e Refinadas armazena soluções válidas e completas. Esta memória é essencial a qualquer *A-Team*, visto que dela serão retiradas as melhores soluções do problema.

Durante a execução de um *A-Team*, pode-se gerar soluções válidas e completas potencialmente boas. Estas soluções podem ter sido geradas por algoritmos de busca global, e, não serem tão boas (refinadas) quanto as soluções presentes na memória de soluções completas e refinadas. Por este motivo, e, dependendo da política de destruição adotada, estas soluções podem ser destruídas antes de serem utilizadas pelos demais algoritmos. Assim, a memória de Soluções Não Refinadas recebe estas soluções que são, via de regra, facilmente aperfeiçoadas por alguma heurística de melhoria. As demais memórias armazenam soluções do tipo especificado pelo seu próprio nome. Embora não esteja especificado na figura 1, a estas memórias podem estar associados outros algoritmos.

Esta estrutura geral tem o propósito de servir como ponto de partida para a especificação de um *A-Team*, visto que nem todos os problemas possuem algoritmos em todas

estas classes e que cada um possui características próprias. Podem existir algoritmos que não se enquadrem na classificação acima. Estes algoritmos podem pertencer a uma combinação das classes mencionadas e não devem ser desconsiderados. A figura 1 foi adotada pela simplicidade de compreensão e não representa todas as possíveis combinações e fluxo dos algoritmos. Esta estrutura deve ser modificada conforme os algoritmos disponíveis e fluxos necessários. A oportunidade de estudar vários algoritmos de um mesmo problema, juntamente ao pensar em como agrupá-los cooperativamente, pode sugerir a idealização de novos algoritmos adequados ao *A-Team*.

Como mencionado anteriormente, a cada memória existe um agente Destruidor. Na estrutura da figura 1 está representado apenas um Destruidor, na memória de Soluções Completas e Refinadas. Embora a política de destruir a pior solução pareça razoável (por estarmos em busca das melhores), esta política pode destruir rapidamente a diversidade entre as soluções, podendo manter o *A-Team* em torno de alguns mínimos locais. Outras políticas de destruição devem ser analisadas e experimentadas. Pode-se destruir as soluções aleatoriamente ou conforme probabilidades associadas às soluções . Estas probabilidades podem ser relativas a alguma característica das soluções (qualidade, número de restrições violadas, tempo em que a solução permaneceu na memória, etc).

As demais memórias, a princípio, não armazenam as soluções que serão utilizadas como resposta final do problema. Por este motivo, podemos considerar que estas memórias funcionam como *buffers* entre os algoritmos que depositam e retiram soluções. Portanto, sempre que uma solução for lida por algum algoritmo, ela é automaticamente destruída. Esta característica também deve ser considerada como sugestão, uma vez que essas memórias podem alcançar outros níveis de complexidade e ser tratadas igualmente à memória de soluções Completas e Refinadas.

Implementação

Devido à total independência entre os agentes dos *A-Teams*, sua implementação pode ser modularizada. Além das vantagens naturais, a implementação modularizada permite uma análise evolutiva do *A-Team*.

Evolução do Time

Se o *A-Team* obtido não produz os resultados especificados anteriormente, é sinal de que são necessárias modificações. Neste caso, qual o comportamento da memória de soluções completas e refinadas? Esta memória pode estar convergindo rapidamente e "estacionar" em mínimos locais. Neste caso, pode-se aumentar o tamanho da memória, introduzir agentes que favoreçam diversidade (p.e. algoritmos aleatórios) ou modificar o(s) destruidor(es). Se as soluçõs da memória são substituídas demasiadamente sem que a qualidade geral seja sensivelmente melhorada, pode-se inserir algoritmos com filosofia de busca local, com o objetivo de explorar a potencialidade de cada solução. Pode-se tentar também a introdução de ciclos que contenham algoritmos de Consenso, a fim de dirigir a busca para alguma região do espaço de soluções.

Se, após exaustivas modificações, os resultados obtidos pelo *A-Team* não satisfazem,

fica a certeza de que nenhum dos métodos aproximado utilizados seria capaz de produzir soluções melhores.

Por serem organizações "abertas", os *A-Teams* garantem-nos, no pior caso, soluções tão boas quanto as soluções fornecidas por métodos aproximados de construção. Para que isso seja verdade, basta introduzir no início da execução do *A-Team* as soluções fornecidas por tais métodos.

4 Um experimento

O *Flow Shop Problem* (FSP) é um problema NP-Difícil clássico de escalonamento de tarefas, que pode ser definido como se segue: um conjunto de n jobs, cada job composto por m tarefas, deve ser processado na mesma seqüência nas m máquinas disponíveis [Bak74]. O objetivo é encontrar uma seqüência de jobs que minimize o *Makespan*.

Utilizando a metodologia proposta e algumas heurísticas da literatura, estes autores propuseram alguns *A-Teams* para o FSP, obtendo resultados extremamente próximos aos melhores valores conhecidos (menos de 0.4% de distâcia). Este trabalho continua em andamento e resultados melhores são esperados. Vale ressaltar que os demais algoritmos, executados isoladamente, não foram capazes de produzir resultados comparáveis aos dos *A-Teams*. A descrição completa deste trabalho pode ser encontrada em [PS94].

5 Conclusões

Este artigo descreveu uma metodologia de especificação de *A-Teams* para problemas de Otimização Combinatória. A motivação deste trabalho decorre do sucesso obtido no uso de *A-Teams* em problemas de grande porte. Os principais conceitos desta nova técnica e um primeiro experimento foram sucintamente abordados.

Referências

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling.* New York, John Wiley, 1974.

[Che92] C. L. Chen. *Baysean Nets and A-Teams for Power System Fault Diagnosis.* PhD thesis, Electrical and Computer Engineering Department, Carnegie Mellow University, Pittsburgh, PA, 1992.

[GJ79] R. M. Garey e D. S. Johnson. *Computers and intractability, a guide to the theory of NP-Completeness.* W.H. Freeman and co., 1979.

[Mur92] S. Murthy. *Synergy in cooperating agents: designing manipulators from task specifications.* PhD thesis, Electrical and Computer Engineering Department, Carnegie Mellow University, Pittsburgh, PA, 1992.

[PS94] H. P. Peixoto e P. S. Souza. Times assíncronous: Uma nova técnica para o flow shop problem. Seminário Integrado de Software e Hardware, Caxambu - MG - Brasil, agosto de 1994.

[Sou93] P. S. Souza. *Asyncronous Organizations for Multi-algorithm Problems*. PhD thesis, Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburg,PA, 1993.

[TS92] S. N. Talukdar e P. S. Souza. Scale efficient organizations. *Proceedings of the 1992 IEEE International Conference on Systems, Man and Cybernetics, Chicago*, outubro de 1992.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o** *Flow Shop Problem*, *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about anoother agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cl'audio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`