Uma arquitetura multiagente para a solução de problemas de seqüenciamento da produção

Carlos Alberto dos Santos Passos[1]
Sara Luísa de Andrade Fonseca[2]

[1]Centro de Pesquisas Renato Archer - CenPRA/MCT
Rod. D. Pedro I, km 143,6 – CEP 13069-901 – Campinas (SP)
carlos.passos@cenpra.gov.br

[2]Faculdade de Engenharia Elétrica e de Computação – FEEC/Unicamp
Cidade Universitária "Zeferino Vaz" - 13083-970 - Campinas (SP)
sfonseca@fee.unicamp.br

Resumo: Este artigo descreve a utilização de uma arquitetura multiagente, baseada em Times Assíncronos (A-Teams) proposto originalmente por Sarosh Talukdar da *Carnegie Mellon University* dos EUA, para resolver problemas de seqüenciamento da produção em *flowshops* com limitação na oferta de recursos de uso compartilhado. Este tipo de produção é bastante comum em Indústrias de Processos Químicos, como por exemplo as dos ramos alimentício e farmacêutico. Este tipo de abordagem pode também ser empregada em outros tipos de problema e outros domínios de aplicação, como por exemplo: problemas de planejamento de produção de médio e longo prazos, planejamento de rotas e controle de tráfego de trens. A metodologia proposta considera a utilização cooperativa e sinérgica dos seguintes algoritmos: Busca Tabu, Algoritmos Genéticos, *Beam-Search* e heurística Cheap-NEH.
Palavras-Chave: Multiagentes, A-Teams, algoritmos heurísticos, seqüenciamento.

A multi-agent architecture to solve sequencing problems

Abstract: This article describes the use of a multi-agent architecture, based in A-Teams, proposed originally by Sarosh Talukdar (Carnegie Mellon University, EUA), to solve sequencing problems in flowshops with limited availability of shared resources. This type of production is very common in Chemical Industries, like foodstuffs and pharmaceutical, for example. This type of approach may be used in other types of problems and in other fields of application, like: problems of production planning on medium and long term, routing planning and train traffic control. The proposed methodology considers the cooperative use of the following algorithms: Tabu Search, Genetic Algorithms, Beam-Search and the heuristic Cheap-NEH.
Keywords: Multi-agents, A-Teams, heuristics, sequencing.

(Received April 02, 2005 / Accepted Junel 18, 2005)

1. Introdução

O modelo *flowshop* de produção é amplamente utilizado em indústrias químicas. A principal característica deste tipo de indústria é a produção em série em instalações pequenas ou médias, com grande variedade de produtos [10]. Isto ocorre devido à familiaridade presente entre os produtos, cuja fabricação exige operações intermediárias que são executadas na mesma ordem, e à tendência de uma organização estática das unidades de produção (máquinas) utilizadas. Além disso, as operações para produção destes produtos geralmente compartilham recursos como vapor de água e eletricidade, que são limitados e impõem uma dificuldade adicional à solução do *flowshop* tradicional.

O modelo *flowshop* é basicamente um fluxo unidirecional de produtos através das unidades de produção. Em outras palavras, o produto flui através da fábrica sempre na mesma direção. Eventualmente, um

produto pode pular um estágio de produção, mas sempre mantendo o mesmo sentido de fluxo.

Uma definição formal de *flowshop* pode ser encontrada em [1]. A fábrica contém *M* máquinas distintas e N tarefas. Cada tarefa consiste de *M* operações, cada uma das quais requer uma máquina diferente. As máquinas em um *flowshop* são numeradas *1, 2, ..., M*; e as operações da tarefa i são correspondentemente numeradas (*i, 1*), (*i, 2*), ..., (*i, M*). Cada tarefa pode ser tratada como se possuísse exatamente *M* operações. Cada operação necessita de um certo tempo para ser processada em cada uma das máquinas, bem como uma determinada quantidade de recursos, que estão limitados de acordo com a sua respectiva oferta máxima instantânea. Quando existem menos operações, o tempo correspondente às operações inexistentes pode ser tomado como zero. O objetivo é sequenciar as operações minimizando o tempo total de produção (*makespan*), respeitando as restrições do problema.

Encontrar a seqüência que minimiza o *makespan*, é um problema NP-difícil [3]. Isto inviabiliza a procura pela solução ótima através de algoritmos exatos e abre caminho para a abordagem do problema através de métodos aproximados, como heurísticas e meta-heurísticas, que procuram encontrar soluções aceitáveis, eventualmente ótimas, em um tempo razoável.

Meta-heurísticas, tais como: *Beam Search* (BS), Busca Tabu (BT), Algoritmos Genéticos (AG) e *Simulated Annealing* (SA), têm sido utilizadas com muito sucesso em problemas de otimização [2]. A principal razão é que este tipo de abordagem apresenta bom desempenho e encontra soluções de boa qualidade em comparação às abordagens convencionais onde se empregam técnicas de otimização clássicas (como programação matemática) ou heurísticas simples, como regras de despacho, por exemplo. Neste sentido, a utilização de A-Teams abre uma nova perspectiva para a resolução do problema de seqüenciamento, pois permite a combinação destas técnicas e a obtenção de soluções potencialmente melhores do que aquelas obtidas por alguma destas técnicas individualmente.

Os autores deste artigo, em trabalhos precedentes, investigaram a utilização de outras abordagens para o mesmo tipo de problema. Entre eles estão: Busca Tabu, Algoritmos Genéticos e *Beam Search* [8], [9] e [7] respectivamente. Cada uma destas abordagens foi experimentada individualmente e, baseando-se nas suas características intrínsecas, apresentam boas soluções para casos particulares.

Este artigo está organizado da seguinte forma: a seção 2 descreve trabalhos correlatos, a seção 3 descreve a metodologia empregada, a seção 4 apresenta os resultados obtidos, a seção 5 apresenta as conclusões e a seção 6 as referências bibliográficas.

2. Trabalhos Correlatos

Diversas meta-heurísticas, como AG e, principalmente, BT, já foram aplicadas ao problema de flowshop e uma vasta literatura pode ser encontrada a este respeito. Em [12] estão listados algumas das técnicas utilizadas com sucesso na busca de soluções factíveis e exemplos de limites teóricos mínimos para efeito de comparação.

Na literatura é possível ainda encontrar várias propostas que combinam técnicas exatas de programação matemática, como *branch and bound*, por exemplo, e técnicas baseadas em métodos heurísticos [10] aplicadas ao problema de seqüenciamento.

A utilização de abordagens baseada em A-Teams para resolver este tipo de problema também foi investigada por vários autores. Exemplos disto podem ser encontrados em [5], [13] e [14].

O trabalho aqui descrito procura explorar a já provada aplicabilidade dos A-Teams e das meta-heuríticas citadas, em uma combinação inovadora e particularmente complexa.

3. Metodologia

A metodologia adotada para a resolução de problemas de seqüenciamento utiliza uma abordagem multiagente baseada na técnica conhecida na literatura com *Asynchronous Teams* ou A-Teams [13]. A técnica de A-Teams foi proposta originalmente por Sarosh Talukdar da *Carnegie Mellon University* - CMU/USA [13] e [14]. A-Teams, como definido por Talukdar, são organizações abertas de alto desempenho para a solução de problemas difíceis, particularmente problemas das áreas de planejamento, projeto, seqüenciamento e controle em tempo real.

A técnica A-Teams utiliza agentes para otimizar um conjunto de soluções, chamado de população de soluções, armazenados em um elemento da arquitetura, denominado de memória. O acesso ou a manipulação destas soluções é realizado de forma assíncrona e independente por cada um dos agentes que compõe o A-Teams. As principais características de um A-Teams são:

- Autonomia dos agentes – os agentes são completamente independentes um dos outros.

- Fluxo de dados cíclicos – o fluxo de dados cíclicos permite o compartilhamento de resultados pelos diferentes agentes possibilitando a cooperação e a obtenção de soluções melhores dos que as obtidas individualmente por cada um deles.
- Comunicação assíncrona – os agentes lêem e escrevem nas memórias sem nenhuma sincronização entre eles.

3.1 Memórias

Memórias em um A-Teams representam uma população de soluções de um único tipo. Elas armazenam soluções candidatas e são manipuladas pelos agentes durante a evolução do algoritmo.

Na arquitetura proposta existem dois tipos de memórias: uma memória contendo soluções completas e factíveis e uma memória contendo soluções parciais.

A primeira armazena apenas soluções consideradas admissíveis para o problema que está sendo resolvido. As soluções nesta memória são armazenadas obedecendo aos valores de seus respectivos *makespan*, visando facilitar o processamento dos agentes, uma vez que todos os agentes utilizam o mesmo critério de desempenho.

A segunda contém soluções com diferentes tamanhos (*tamanho* $\leq N$) e normalmente tem algumas posições vazias na seqüência que elas representam. Como estas soluções não permitem o cálculo da função de custo – *makespan* no caso – o processo de seleção é baseado na regra FIFO (*First-In-First-Out)*.

3.2 Agentes

A-Teams utilizam agentes para otimizar a população de soluções e cooperam compartilhando o acesso às populações de soluções candidatas armazenadas nas memórias. Cada agente encapsula um método particular de solução do problema que age modificando as soluções existentes.

Os agentes em um A-Teams podem ser classificados basicamente como sendo de três tipos: agentes de construção, de melhoria e de destruição. Agentes de construção são aplicados à definição do problema criando novas soluções e inicializando a memória de soluções. Agentes de melhoria procuram aumentar a qualidade das soluções, aplicando algoritmos próprios às soluções existentes. Finalmente, agentes de destruição têm o papel de limitar o número de soluções na memória, eliminando soluções ruins e direcionando a população de soluções em direção às boas soluções [5].

3.3 Abordagem proposta

Esta seção apresenta a arquitetura proposta para o A-Teams, descreve a sua estrutura de fluxo de dados e apresenta os agentes de melhoria e os agentes de destruição. O agente de construção é implementado utilizando um algoritmo do tipo *Beam Search* [7].

3.3.1 Arquitetura A-Teams e estrutura de fluxo de dados

No A-Teams proposto há duas memórias - uma para soluções completas e outra para soluções parciais, memórias principal e parcial respectivamente - um agente de destruição (DE), um agente de construção (CC), agentes de melhoria que lêem e escrevem na memória principal: Busca Tabu (BT), Algoritmo Genético (AG) e um mecanismo adicional que combina dois agentes agindo entre a memória principal e a memória parcial (C e Cheap-NEH). Os agentes C e Cheap-NEH operam entre as duas memórias criando um ciclo de desconstrução e construção de soluções factíveis.

A Figura 1 mostra a arquitetura implementada com a estrutura do fluxo de dados (setas), os agentes (CC, BT, AG, C, Cheap-NEH e DE) e as memórias (retângulos).

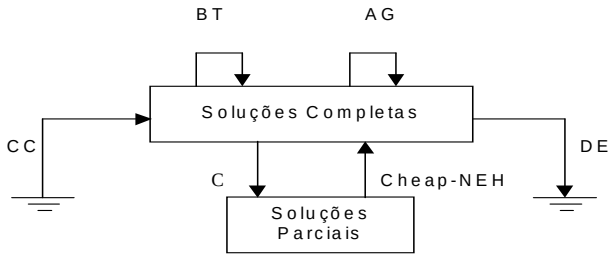

Figura 1: Arquitetura do A-Teams proposto

3.4 Agentes de melhoria

Os agentes de melhoria da arquitetura proposta, conforme mencionado anteriormente, são de três tipos: Busca Tabu, Algoritmo Genético e um mecanismo adicional combinando dois agentes que agem entre a memória principal e a memória parcial, os algoritmos C e Cheap-NEH. Estes algoritmos são apresentados a seguir.

3.4.1 Algoritmo Genético

Os algoritmos genéticos foram introduzidos por Holland em 1975 [4], para explicar o desenvolvimento de sistemas artificiais que retêm os mecanismos naturais de adaptação.

Eles são utilizados para aplicar evolução a uma população de soluções e permitir a otimização de uma medida de desempenho destas soluções, a função-objetivo. Cada solução para o problema em questão é um indivíduo (ou cromossomo) da população. Sua adaptação é medida pela função-objetivo (*makespan*) que representa, então, a pressão do ambiente. Através desta medida, indivíduos são selecionados para reprodução e operadores como recombinação e mutação são aplicados a estes indivíduos. O algoritmo de uma forma simplificada pode ser descrito como:

Inicializar a população de cromossomos.
Repita até atender ao critério de parada
    Avaliar cada cromossomo (função-objetivo).
    Selecionar e Aplicar Operadores (Recombinação, Mutação).
Fim-Repita
Retornar o melhor cromossomo.

Ao interagir com o A-Teams, o AG, a cada certo número de gerações, incorpora em sua população algumas soluções da memória de soluções completas e escreve o seu melhor resultado obtido até então. Uma alternativa para aplicação deste agente, também considerada, é reinicializar o processo evolutivo a cada vez que lê novas soluções da memória, incorporando as soluções na população inicial.

Levando em conta o fato do AG não ser um agente determinístico [4], foram propostas as seguintes políticas de leitura:

- Selecionar as melhores soluções presentes na memória, segundo o *makespan*.
- Selecionar aleatoriamente um conjunto de soluções.
- Selecionar soluções da melhor para a pior, com probabilidade linear de escolha (a probabilidade de escolha é inversamente proporcional ao *makespan*).
- Selecionar soluções da pior para a melhor com probabilidade linear de escolha (a probabilidade de escolha é proporcional ao *makespan*).

Os detalhes a respeito da implementação deste agente podem ser encontrados em [9]. Esta referência mostra como os diversos parâmetros inerentes a um AG foram testados. Na representação utilizada, cada indivíduo é uma seqüência de números inteiros representando cada uma das tarefas.

Foram testados dois tipos de recombinação (OX e uniforme) além de três tipos de seleção de pais (aleatória, bi-classista e roleta). Recombinação do tipo OX é aquela do tipo simples, com substituição de eventuais elementos repetidos. Na Recombinação uniforme, cada elemento componente do indivíduo tem a mesma chance de vir de um dos pais.

Os melhores resultados foram obtidos utilizando-se recombinação uniforme aliado à seleção bi-classista. Nesta seleção, um dos pais é selecionado entre os melhores indivíduos da população e o outro entre os piores. Também foram testadas diversas taxas de recombinação e de mutação. Foi encontrada uma solução de compromisso: 50% e 0 a 15%, respectivamente. Exemplificando-se a representação utilizada aplicando-se o operador de recombinação uniforme, temos: **4** 3 **2** 1 + 2 **1** 4 **3** → 4 1 2 3 (Tarefa 2, seguida de 1, 4 e 3).

No A-Teams aqui proposto, o AG trabalha sobre a memória de soluções completas, funcionando como um agente de melhoria. O AG utiliza os parâmetros descritos acima, além daqueles que tiveram que ser especialmente refinados para o A-Team e que estão descritos na seção de resultados.

3.4.2 Busca Tabu

Glover em [2] descreve a Busca Tabu como uma busca na vizinhança com uma lista de posições visitadas recentemente. A Busca Tabu pode ser vista como um método de minimização de uma função objetivo *F*. A busca parte de uma solução inicial *S* e procura uma solução *S'* em sua vizinhança. Se $F(S') < F(S)$, então *S* recebe o valor de *S'* e a busca é reiniciada. Podendo-se adotar vários critérios de parada.

No entanto, a mudança de *S* para *S'* pode gerar ciclos entre as soluções. Para que isto seja evitado é construída uma lista Tabu para restringir as mudanças. Estas restrições não agem isoladamente, sendo complementadas por critérios de aspiração, que permitem às posições que estão na lista serem aceitas. A condição de parada pode variar de acordo com os critérios adotados, mas normalmente o algoritmo pára após um determinado número de iterações. O algoritmo de Busca Tabu pode ser descrito como abaixo:

Selecione uma solução inicial *S*, faça dela a melhor solução: $S^*=S$ e $F^*=F(S)$.
Repita até que seja atingido um critério de parada
Encontre uma solução *S'* pertencente à vizinhança de *S* (V_S), onde $F(S^*)$ seja um mínimo em V_S
    Se a mudança de *S* para *S'* não é Tabu
        Então Se $F(S') < F^*$,
            $S^*=S$, $F^*=F(S)$ e

Fim Então
Faça $S=S'$ e atualize a lista Tabu
Fim Se
Se não a melhor solução em V_S será fixada
Fim-Repita

Os critérios de parada adotados e detalhes da implementação deste agente podem ser encontrados em [8]. No A-Teams aqui proposto, a BT, assim como o AG, funciona como um agente de melhoria, tomando como solução inicial uma das soluções da memória principal.

Como a BT tende a atingir resultados iguais para uma mesma solução inicial, em todos os protocolos de leitura não se permite que se repita a leitura de uma solução. Foram propostas as seguintes políticas:

- Selecionar soluções da melhor para a pior com probabilidade linear de escolha (a probabilidade de escolha é inversamente proporcional ao *makespan*).
- Selecionar soluções da pior para a melhor com probabilidade linear de escolha (a probabilidade de escolha é proporcional ao *makespan*).
- Selecionar a melhor solução ainda não lida.

3.4.3 Heurística Cheap-NEH

Cheap-NEH é uma heurística não-determinística [6] de construção, que a partir de uma solução parcial constrói uma solução completa adicionando novas tarefas às posições vazias. Trata-se de uma derivação da heurística determinística NEH proposta em 1983 por Nawaz et al. em [6] para o problema *flowshop* de permutação.

A Cheap-NEH inicia seu processamento construindo inicialmente um conjunto E contendo as tarefas não presentes na solução parcial. Em seguida, encontra a primeira posição vazia na seqüência parcial. Escolhe então, aleatoriamente uma das tarefas do conjunto E e completa a seqüência até a posição em questão, alternando a posição da tarefa e verificando em que posição o *makespan* desta seqüência parcial é menor. Prossegue desta forma completando a seqüência. O algoritmo abaixo descreve o funcionamento da heurística:

Construa um conjunto E com as tarefas não presentes na seqüência parcial
Para $p=1$ até N **faça**
Se posição p da seqüência parcial não possui tarefa
Então Escolha aleatoriamente uma tarefa i em E
Insira a tarefa i, entre as p possibilidades, na posição que minimize o *makespan* parcial.

3.4.4 Agente de destruição

O agente de destruição (DE) atua no momento da inserção de novas soluções na memória de soluções completas. Este agente aceita ou não uma nova solução apresentada por um dos outros agentes segundo uma das seguintes políticas de aceitação:

- Aceita qualquer nova solução.
- Aceita apenas soluções que possuam *makespan* melhor do que a pior já presente na memória.

Para selecionar a solução a ser retirada da memória, utiliza uma das seguintes políticas de destruição:

- Destruição da pior solução: a pior solução é destruída com probabilidade 1.
- Destruição segundo probabilidade uniforme: todas as soluções têm a mesma probabilidade de serem destruídas, exceto a melhor, que tem probabilidade 0.
- Destruição segundo probabilidade linear: cada solução tem uma probabilidade de ser destruída que é inversamente proporcional ao *makespan*. A melhor solução tem probabilidade 0.

4. Resultados

A versão atual do programa que implementa a abordagem multiagente está sendo executada em um computador com um único processador e em uma versão seqüencial do mesmo. Nesta versão, cada agente tem a mesma probabilidade de ser executado em um determinado instante. A fim avaliar a solução proposta e as respectivas implementações dos algoritmos, dois casos do estudo foram realizados:

i) uma instância do problema com 100 tarefas e 20 máquinas com restrição nos recursos de uso compartilhados, para a qual foi aplicada a abordagem multiagente com A-Teams e os algoritmos de Busca Tabu e Algoritmos Genético separadamente para comparação entre eles; e
ii) um conjunto de instâncias de problemas do tipo *flowshop* obtido na literatura, conhecidos como instâncias de Taillard [11]. Estas instâncias possuem um número de tarefas entre 20 e 500 e um número de máquinas entre 5 e 20 e não possuem restrição nos recursos de uso compartilhado.

A primeira etapa dos testes consiste na determinação dos valores dos parâmetros a serem usados nos agentes e na seleção da combinação de políticas aplicadas aos algoritmos. Na determinação do tamanho da memória

completa das soluções, os testes indicaram que os melhores resultados são obtidos com um tamanho de *2*N* soluções, onde *N* é o número de tarefas do problema.

Os parâmetros dos agentes para o algoritmo de BT foram ajustados como apresentados na Tabela 1, e para o AG como apresentado na Tabela 2. Eles são diretamente proporcionais à raiz quadrada de *N*. Os testes realizados mostraram que esta opção é boa o bastante para controlar o aumento do tempo de execução em função de *N* quando comparada com a utilização de uma função linear de *N*. Os valores adotados para o critério de parada de cada algoritmo representam um compromisso entre o tempo para a execução de cada agente e o tempo total de execução do A-Teams. Este valor deve permitir que cada agente atue na melhoria das soluções e que exista uma efetiva cooperação entre eles.

O AG apresentou melhores resultados quando executado com uma granularidade pequena, ou seja, executando durante um número pequeno de gerações antes de escrever e ler novamente da memória de soluções completas. Entretanto, este número de gerações não pode ser tão pequeno a ponto de impossibilitar uma real evolução do algoritmo e uma contribuição do mesmo para busca de melhores soluções.

Uma alternativa testada para este agente de melhoria é reinicializá-lo a cada escrita/leitura na memória. Neste caso, há a necessidade de uma granularidade maior, visto que toda sua evolução deve acontecer em apenas um ciclo do A-Teams. A última opção de configuração para o AG apresentou resultados mais satisfatórios e foi utilizada para a obtenção dos resultados aqui apresentados.

Os diversos testes realizados com o algoritmo de A-Teams mostraram que o melhor conjunto de políticas, dentre as propostas é:

- Política de Leitura da BT: Da pior para a melhor com distribuição linear de probabilidade de escolha.
- Política de Leitura do AG: Da melhor para a pior com probabilidade uniforme de escolha.
- Política de Aceitação: Aceitação de qualquer nova solução.
- Política de Destruição: Destruição de soluções com probabilidade uniforme, a menos da melhor solução.
- Consenso por Intersecção

Parâmetro	**Valor**
FIX*	100* sqrt(N)
NO_IMPRV*$_1$	10 * sqrt(N)
PERCENT*$_2$	86

Tabela 1**:** Parâmetro da Busca Tabu (Critérios de parada)

(*) número total de interações; (*1) número total de iterações sem melhora da função objetivo; (*2) porcentagem de novas seqüências geradas em um passo que já estão presentes na lista Tabu existente.

Parâmetro	**Valor**
Tamanho da População	60* sqrt(N)
Numero de Gerações (critério de parada)	120 * sqrt(N)

Tabela 2**:** Parâmetros do Algoritmo Genético

Obs: outros parâmetros do AG são aqueles descritos na seção 3.4.1

Pode-se notar que as políticas dos dois agentes de melhoria são complementares, fazendo com que cada um trabalhe uma parte das soluções da memória de soluções completas. A Figura 2 a seguir apresenta uma comparação entre o desempenho das duas heurísticas de melhoria utilizadas separadamente na resolução do problema e do A-Teams adotado, utilizando o conjunto de políticas descrito anteriormente. Utilizou-se uma instância de testes de 100 tarefas e 20 máquinas. Destaca-se o comportamento sinérgico resultante da cooperação entre os algoritmos, já que os resultados apresentados pelo A-Teams superaram os resultados apresentados pelas outras heurísticas quando utilizadas separadamente. O mesmo comportamento do algoritmo foi encontrado em todas as experiências executadas com diversos problemas de diferentes tamanhos.

A abordagem utilizando A-Teams foi testada também com um conjunto de exemplos obtidos na literatura, proposto originalmente por Taillard [11]. Todos os exemplos podem ser encontrados em [12].

A Tabela 3 apresenta as doze instâncias selecionadas das 120 apresentadas por Taillard. Elas foram selecionadas para cobrir problemas de diferentes tamanho com o número de tarefas (*jobs*) variando de 20 a 500 e o número de processadores (máquinas) variando de 5 a 20. Apenas uma instância para problemas de um mesmo tamanho foi selecionada. Estes valores cobrem a maioria dos problemas práticos existentes na indústria de processos químicos. Na Tabela 3, as instâncias aparecem como taxxx (*N x M*), onde xxx é o número do problema, *N* é a quantidade de tarefas e *M* é a

quantidade de processadores. A segunda coluna desta tabela indica a diferença percentual média entre o limite mínimo teórico de cada instância e o resultados de três execuções do A-Teams. A terceira coluna mostra esta mesma diferença para as melhores soluções encontradas na literatura por diferentes algoritmos. Elas são iguais quando a solução ótima da instância é conhecida.

Uma análise desta tabela mostra que embora a abordagem multiagente utilizando A-Teams aqui proposta tenha sido desenvolvida para problemas do tipo *flowshop* com limitação no consumo de recursos compartilhados de produção, pode-se verificar por intermédio da utilização das instâncias de Taillard que ele também apresenta bom desempenho em termos de qualidade de solução e de tempo de execução para problemas clássicos de *flowshops*. Em quase todas as instâncias, o makespan encontrado pelo A-Team proposto está a menos de 5% do melhor valor encontrado na literatura. O tempo de execução cresce expressivamente à medida que cresce o tamanho das instâncias, como poder-se-ia esperar de um problema de natureza combinatorial. É importante notar ainda que: i) os resultados foram obtidos com limitação no tempo de execução, ou seja, com um tempo maior de execução os resultados obtidos poderiam ser ainda melhores; ii) como as instâncias de Taillard são genéricas, i.e., não são necessariamente do mesmo domínio de aplicação aqui tratado, não é esperado que a abordagem multiagente proposta neste trabalho apresente bom desempenho para todas as instâncias e iii) as melhores soluções obtidas para estas instâncias não foram obtidos por um único algoritmo.

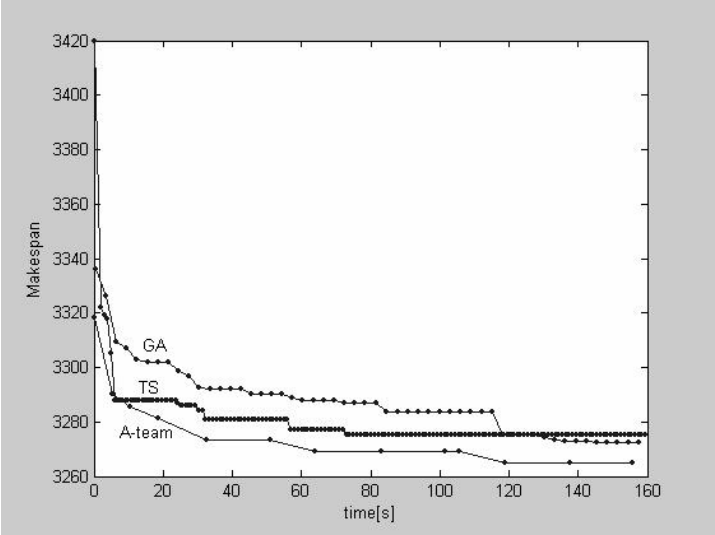

Figura 2: Comparação entre Busca Tabu, Algoritmo Genético e A-Teams

Instância (NxM)	Distância do Limite Mínimo (%)	Distância da Melhor Encontrada(%)	Tempo (s)
ta001 (20 x 5)	0	0	3,5
ta011 (20 x 10)	1,3	1,3	5,6
ta021 (20 x 20)	1,4	1,4	9,4
ta031 (50 x 5)	0	0	21,6
ta041 (50 x 10)	3,4	3,4	38,1
ta051 (50 x 20)	6,2	4	64
ta061 (100 x 5)	0	0	90,5
ta071 (100 x 10)	0,7	0,7	151,2
ta081 (100 x 20)	6,6	5,1	270,3
ta091 (200 x 10)	0,6	0,6	624,4
ta101 (200 x 20)	3,6	3,2	1.125,5
ta111 (500 x 20)	1,8	1,7	11.223,1

Tabela 3: Resultados das instâncias de *Benchmark*

5. Conclusões

Este artigo descreve a aplicação de uma arquitetura baseada em agentes, chamada de A-Teams, na solução de problemas de *flowshop* com limitação na oferta de recursos de uso compartilhado. A principal característica é a cooperação entre os diferentes agentes (algoritmos) de uma forma autônoma e assíncrona. Esta cooperação ocorre através do compartilhamento de soluções presentes nas memórias.

A utilização dos agentes baseados nos algoritmos de Busca Tabu e Genético, em conjunto com os demais agentes, possibilitou que as características de cada um deles fossem combinadas permitindo a obtenção de resultados melhores do que os obtidos individualmente.

Os casos de testes aplicados ao algoritmo multiagente mostraram que os algoritmos cooperam entre si apresentando resultados em termos de qualidade de solução e desempenho computacional superiores aos obtidos anteriormente com os algoritmos utilizados isoladamente.

Os resultados apresentados quando comparados com as instâncias de Taillard, que servem de referencial comparativo (*benchmarks)*, situam o algoritmo desenvolvido em relação a outras técnicas propostas na literatura, e mostram resultados bastante razoáveis para a maioria das instâncias.

A abordagem multiagente baseada na técnica de A-Teams aqui proposta possibilita a utilização de diferentes algoritmos de forma cooperativa e sinérgica e pode ser utilizada para resolver um elenco importante de problemas de otimização combinatorial.

6. Referências Bibliográficas

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

[2] Glover, F.W. & Kochenberger, G.A. *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2002.

[3] Garey M.R., Johnson D.S. *Computers and Intractability; a Guide to NP-Completeness*, New York, W. H. Freemon and Company, 1979, 340 p.

[4] Holland, J.H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[5] Murthy S.; Rachlin J.; Akkiraju R.; Wu F. Agent-Based Cooperative Scheduling; In Constraints & Agents, *Technical Report WS 1997-97-05*, Menlo Park, AAAI Press, 1997.

[6] Nawaz M., Enscore E. E., Ham I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem, OMEGA, *International Journal of Management Science*, 11(1):91-95, 1983.

[7] Passos C.A.S. "A Beam Search Algorithm to Solve *Flowshop* Scheduling Problems with Constraints on Shared Resource". *Anais do IFAC/IFIP/IEEE MCPL'2000 Conference*, Grenoble/França, 2000.

[8] Passos C.A.S.; Nazareth L.G.P. '*Flowshop* Scheduling in Chemical Process Industries using Tabu Search Algorithm'; *Anais do 18ª International Conference on CAD/CAM and Factories of the Future*, INESC/Porto, Portugal, 2002.

[9] Passos C. A. S.; Fonseca S. L. A. 'Scheduling of Jobs in Chemical Process Industries Using Metaheuristics Approaches'; *Anais do International Conference on Industrial Logistics - ICIL'2003* 16, Vaasa, Finlandia, 2003.

[10] Rodrigues, M. T. M. *Sequenciamento e Alocação de Operações em Flow Shops na Indústria Química com Restrições sobre os Recursos Compartilhados. Uma Abordagem de Busca Orientada por Restrições*, Tese de doutoramento, Unicamp, 1992.

[11] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, p. 278-285,1993.

[12] Taillard, E. Problem instances em http: //ina.eivd.ch/collaborateurs/etd/default.htm

[13] Talukdar S.N.; Souza P.S. "Asynchronous Teams." *Anais do Second SIAM Conference on Linear Algebra: Signals, Systems and Control*, 1990.

[14] Talukdar S.N.; Souza P.S. "Scale efficient organizations". *Anais do IEEE Int. Conference on Systems, Man and Cybernetics*, 1992.