Balanceamento de Carga na Paralelização da Meta-heurística GRASP

Adriana Cesário de Faria Alvim e *Celso Carneiro Ribeiro*

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente 225
Rio de Janeiro - 22453-900
E-mail: {alvim, celso}@inf.puc-rio.br

Palavras-chave: Algoritmos paralelos, balanceamento de carga, meta-heurísticas, GRASP, avaliação de desempenho, otimização combinatória.

Resumo: Este trabalho consiste na comparação do desempenho de duas estratégias de paralelização da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*). GRASP caracteriza-se pela independência dos resultados obtidos em cada iteração, o que conduz, naturalmente, a um paralelismo de dados como forma de particionamento do problema. A diferença entre as duas estratégias está na técnica de decomposição dos dados utilizada: *pré-escalonada* (balanceamento estático) ou *auto-escalonada* (balanceamento dinâmico). As duas estratégias foram testadas sob a influência dos mesmos fatores externos, considerando uma aplicação ao problema de alocação ótima de tráfego em sistemas de satélites TDMA. A medida de desempenho considerada foi o tempo marcado pelo relógio (*wall-clock*) utilizado por estratégia. A análise dos resultados computacionais foi feita sobre uma amostra de 180 observações. Diferentes análises apontaram o balanceamento de carga dinâmico como a melhor alternativa para a divisão de trabalho do GRASP paralelo, mostrando significativas melhorias no desempenho da paralelização.

Abstract: We compare two parallelization strategies for the GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic. GRASP is characterized by the independence of its iterations, which naturally leads to data parallelism as the most direct partitioning strategy. The difference between the two parallelization strategies concerns the way in which data is partitioned: pre-scheduled (static load balancing) or self-scheduling (dynamic load balancing). Both strategies were tested under the influence of the same external factors, considering an application to optimal traffic assignment in TDMA satellite systems. The performance of the strategies are compared in terms of the observed elapsed (wall-clock) times on a sample of 180 observations. Dynamic load balancing is shown to be the most suitable strategy for the parallelization of GRASP, with significant improvements in terms of elapsed times.

1. Introdução

A meta-heurística GRASP é uma técnica iterativa de amostragem que vem sendo usada com sucesso para resolver problemas de otimização combinatória (e.g. [6, 12, 13]). Como suas iterações são independentes, sua implementação paralela é imediata: as iterações são distribuídas entre os processadores. Esta técnica de decomposição de domínio (iterações

GRASP) é conhecida por *decomposição por bloco pré-escalonada* [5]. Considerando-se o tempo marcado pelo relógio como a medida de desempenho, esta estratégia pode não ser a mais eficiente em um ambiente de programação paralela com processadores não dedicados. Uma segunda estratégia de paralelização utiliza como técnica de decomposição de domínio o *auto-escalonamento* (ou bolsa de tarefas) [5]. Estabelece-se que um determinado número de iterações representam uma tarefa e então o número de tarefas da *bolsa de tarefas* é determinado pelo número de iterações GRASP. Um único processador *mestre* é responsável por atribuir tarefas aos demais processadores *escravos* e controlar os resultados obtidos. Na primeira estratégia o balanceamento de carga é estático, uma vez que a divisão de trabalho é determinada em tempo de compilação, enquanto na segunda o balanceamento é dinâmico e a divisão de trabalho é determinada em tempo de execução. A eficácia do balanceamento de carga da paralelização da meta-heurística GRASP é estudada comparando-se o desempenho destas duas estratégias. Para sua avaliação, as duas estratégias são executadas simultaneamente, sujeitas à influência dos mesmos fatores externos e usando cada uma o mesmo número de processadores.

Na Seção 2 apresentam-se os conceitos de GRASP. As estratégias de paralelização propostas são discutidas na Seção 3. Na Seção 4 o estudo comparativo das duas estratégias é detalhado. A Seção 5 é dedicada aos resultados computacionais e à análise do experimento. Considerações finais são feitas na Seção 6.

2. GRASP

GRASP [6] é uma técnica iterativa de amostragem randômica, onde em cada iteração obtém-se uma solução para o problema. Cada iteração consiste de duas fases: construção e busca local. Na primeira, uma solução viável é construída, cuja vizinhança será explorada na fase de busca local. A melhor solução entre todas as iterações é guardada como o resultado final.

Na fase de construção, uma solução viável é construída elemento a elemento. Cada elemento a ser incluído na solução é escolhido utilizando-se uma função gulosa adaptativa, que estima o benefício de cada elemento e cria uma lista restrita de candidatos. Seleciona-se então, aleatoriamente, um elemento da lista. A função é adaptativa em razão da lista restrita de candidatos ser recriada a cada iteração da fase de construção, a fim de refletir as mudanças associadas à escolha do elemento anterior. A componente probabilística do GRASP é, portanto, caracterizada pela escolha randômica de um dos melhores candidatos da lista, que não é necessariamente o melhor deles. Desta forma, é possível obter uma grande diversidade de soluções viáveis ao longo das iterações.

A solução gerada pelo GRASP na fase de construção não é necessariamente um ótimo local, mesmo levando-se em consideração definições simples de vizinhanças. Em conseqüência, é quase sempre vantajoso aplicar um procedimento de busca local a fim de melhorar cada solução encontrada na fase de construção. Algoritmos de busca local funcionam de maneira iterativa, trocando sucessivamente a solução corrente por uma solução melhor, encontrada ao se explorar sua vizinhança. A busca local termina ao atingir um ótimo local, quando não encontra melhores soluções na vizinhança da solução corrente. O sucesso do algoritmo de busca local depende da escolha de uma vizinhança adequada, das técnicas de busca em vizinhança utilizadas e da solução inicial. Com respeito a este último ponto, a fase

de construção do GRASP possui um papel importante, uma vez que produz boas soluções iniciais para o procedimento de busca local.

Assim, GRASP pode ser visto como uma meta-heurística que se utiliza de boas características tanto de algoritmos gulosos (soluções de qualidade) como de procedimentos construtivos randomizados (diversificação). O pseudocódigo da Figura 1 ilustra uma implementação GRASP genérica. Como parâmetros de entrada, GRASP utiliza o tamanho máximo (ou o limite para a qualidade dos elementos) da lista restrita de candidatos, o número de iterações e a semente para o gerador de números pseudo-aleatórios. Após ler os dados de entrada na linha 1 e fazer as inicializações na linha 2, as iterações do GRASP são realizadas no *loop* que vai da linha 3 à linha 7. Cada iteração consiste da fase de construção (linha 4), da fase de busca local (linha 5) e, se houver necessidade, da atualização da melhor solução encontrada (linha 6).

```
procedimento GRASP (QualidadeLista, NúmeroIterações, Semente)
1   Ler dados de entrada.
2   Inicializar o custo da melhor solução, Custo(S*) ← ∞.
3   para k = 1,…, NúmeroIterações faça
4       Construir uma solução S de forma randomicamente gulosa.
5       Aplicar um procedimento de busca local em S retornando S̄ .
6       Se Custo(S̄ ) < Custo(S*) então atualizar S* ← S̄ .
7   fim-para
8   Retornar a melhor solução S*.
fim GRASP.
```

Figura 1: Pseudocódigo do procedimento GRASP.

3. Paralelização do GRASP

No modelo de programação distribuída utilizado, uma execução compreende um ou mais processos que se comunicam por troca de mensagens. Um número fixo de processos é criado na inicialização do programa e um único processo é criado por cada processador.

Devido à natureza iterativa do GRASP, o paralelismo de dados é a forma ideal de particionamento do problema. Para decompor os dados, a técnica escolhida deve ter como meta balancear a carga de trabalho e fazer com que a comunicação entre processadores seja mínima e regular [4, 5]. Levando-se em conta que cada iteração requer aproximadamente a mesma quantidade de computação (carga de trabalho regular) e que cada iteração GRASP é independente das demais, cada processador deve executar o mesmo número de iterações. A divisão de trabalho é determinada em tempo de compilação e portanto o balanceamento de carga é estático. A implementação desta técnica, *decomposição por bloco pré-escalonada* [5], em um ambiente multi-processador com p processadores é simples e direta [10]. Cada processador deve receber uma cópia do procedimento, os dados de entrada e uma seqüência independente de números pseudo-aleatórios (é importante assegurar que não haja interseção entre as p seqüências [11]). As K iterações GRASP são divididas entre os p processadores. Cada um dos processadores executa, independentemente e em paralelo, K/p iterações do

procedimento GRASP. A melhor solução dentre as melhores encontradas pelos processadores é identificada como a solução encontrada pelo algoritmo. O custo de comunicação é baixo, uma vez que uma única variável global (a melhor solução) é necessária.

Nesta implementação, todos os processadores executam o mesmo código e realizam as mesmas operações de comunicação. Assim sendo, pode-se dizer que, as cargas de comunicação e de computação dos processadores são praticamente uniformes. Além disto, como mencionado anteriormente, a sobrecarga de comunicação é mínima, uma vez que cada procedimento GRASP trabalha independentemente e uma única operação de comunicação é realizada. Tal paralelismo, inerente à meta-heurística GRASP, conduz, naturalmente, a uma implementação paralela eficiente. O procedimento `GRASPIteraçãoParalela`, descrito com maiores detalhes em Alvim [1], implementou esta estratégia para resolver problemas de alocação ótima de tráfego em sistemas de satélites TDMA. Foi possível observar aceleração linear utilizando-se até 7 processadores dedicados em uma máquina paralela IBM SP2.

No entanto, embora a carga de trabalho assim distribuída seja uniforme, a carga computacional dos processadores envolvidos pode ser bastante heterogênea. Se algum processador estiver mais carregado do que os outros, estes últimos terminarão antes e ficarão ociosos até que todos os processadores terminem suas tarefas. Neste caso, existe uma indicação de que a carga está distribuída desigualmente entre os processadores e que a aplicação poderia ser executada em menos tempo.

Outra técnica de decomposição de domínio conhecida como *auto-escalonamento* (ou bolsa de tarefas) [4] é uma alternativa para lidar com o problema de desbalanceamento de carga descrito acima. Um único processador *mestre* coordena o trabalho realizado por todos os processadores *escravos*. Determina-se que um certo número *bloco* de iterações representam uma tarefa e que existem *K*/*bloco* tarefas. Inicialmente, o *mestre* atribui uma tarefa para cada *escravo*. A medida em que os *escravos* terminam suas tarefas, comunicam tais eventos ao *mestre* e solicitam mais trabalho. Eventualmente, alguns *escravos* vão completar mais tarefas no mesmo tempo em que outro *escravo* levará para terminar sua tarefa inicial. Assim, todos os processadores permanecem ocupados, o tempo ocioso de cada processador é reduzido e o tempo necessário para completar o trabalho (as *K*/*bloco* tarefas) também. A contrapartida é que, conforme o número de processadores aumenta, também aumenta o tempo necessário para avaliar e coordenar os resultados. Pode-se criar um gargalo, com *escravos* ociosos esperando resposta do *mestre*. O algoritmo `GRASPMestreEscravo`, especificado em Alvim [1], usa esta técnica para a paralelização do GRASP.

Diferentemente do procedimento `GRASPIteraçãoParalela`, as cargas de comunicação e trabalho realizadas pelos processadores *escravos* no procedimento `GRASPMestreEscravo` não são necessariamente uniformes. Enquanto na primeira estratégia a carga de trabalho realizada por cada processador é sempre a mesma (exatamente *K*/*p* iterações), esta carga pode variar muito para os processadores *escravos* na segunda. O número de operações de comunicação também não é regular. No entanto, o tamanho da mensagem enviada por um processador *escravo* para comunicar o fim de uma tarefa ao *mestre* é mínimo, constituindo-se de um único dígito. O *escravo* só precisa enviar a melhor solução por ele encontrada uma única vez, quando o *mestre* comunicar que não existem mais tarefas a serem realizadas.

A solução apresentada pelo algoritmo `GRASPIteraçãoParalela` usa o modelo de programação paralela SPMD com balanceamento de carga estático, ao utilizar a técnica de

decomposição de domínio pré-escalonada. O algoritmo `GRASPMestreEscravo` apresenta uma solução SPMD (todos os processadores executam o mesmo código operando em diferentes dados) com o modelo Mestre/Escravo embutido (o fluxo de controle atribui uma certa tarefa para o *mestre* ou *escravo*) e realiza um balanceamento de carga dinâmico, pois as tarefas atribuídas aos processadores são determinadas em tempo de execução.

4. Balanceamento estático versus dinâmico

Nesta seção, discute-se a metodologia empregada para comparar o desempenho das duas estratégias de balanceamento de carga da meta-heurística GRASP apresentadas anteriormente: estático (procedimento `GRASPIteraçãoParalela`) e dinâmico (procedimento `GRASPMestreEscravo`). Como um dos principais objetivos do processamento paralelo é a redução do tempo real (*wall-clock*) ou do custo necessário para completar um trabalho [3], o tempo marcado pelo relógio será a medida de performance utilizada nos experimentos.

Um experimento computacional que compara o desempenho de dois algoritmos para uma mesma classe de problemas deve levar em consideração diferentes fatores que afetam os resultados dos testes. Mesmo controlando-se diversos fatores do ambiente [2], tais como *hardware*, *software* e os problemas-teste utilizados, a repetição de um mesmo experimento pode levar a resultados muito variados. Um fator difícil de ser controlado é a influência de outros programas que competem por recursos compartilhados, tais como os processadores, a rede e leitura/gravação de dados. Uma forma de reduzir a variabilidade dos resultados é executar os experimentos em máquinas dedicadas ou em períodos de cargas leves e uniformes. Porém, quando se deseja comparar a performance de dois algoritmos, tomando-se como medida de desempenho o melhor tempo, é desejável que a carga de trabalho do ambiente computacional dos testes seja representativa do ambiente real onde os algoritmos serão utilizados. Observa-se ainda que a carga real não se repete, nem é reprodutível.

Ao invés de tentar controlar a carga do sistema, uma idéia consiste em forçar as duas implementações a serem testadas sob a influência dos mesmos fatores externos. A avaliação do desempenho relativo de cada estratégia pode ser realizada executando-se as duas aplicações simultaneamente, cada uma com o mesmo número de processadores, porém utilizando diferentes processadores. Usando-se processadores diferentes, as duas aplicações não disputam recursos compartilhados entre si, exceto em termos de comunicação através da rede (o que, dependendo da aplicação, pode ser de menor importância, como ocorre no GRASP devido ao reduzido número de mensagens). Além disso, executando-se diversas vezes as duas aplicações simultaneamente, em processadores selecionados aleatoriamente, na média a carga externa que afeta cada performance tende a ser a mesma. Pode-se assim fazer a comparação de dois algoritmos aplicados a um mesmo problema com base em seu desempenho médio relativo, executando-os em um mesmo sistema, utilizando as mesmas máquinas e submetidos a cargas externas muito similares.

A implementação desta metodologia é feita de forma bastante simples. Um número par igual a $2p$ processadores é alocado para a execução simultânea das duas estratégias de balanceamento, cada uma utilizando metade dos processadores. Um processador é eleito para repartir aleatoriamente os $2p$ processadores entre as duas estratégias, de modo a evitar distorções provocadas pela alocação sistemática de um mesmo processador à mesma estratégia. Os dois procedimentos são portanto executados simultaneamente, sob a influência

da mesma carga externa. Enquanto o procedimento `GRASPIteraçãoParalela` utiliza efetivamente seus *p* processadores para a realização de *K*/*p* iterações GRASP cada um, o procedimento `GRASPMestreEscravo` utiliza apenas *p* - 1 processadores (*escravos*) para a execução destas iterações, reservando um para atuar como *mestre*, distribuindo as tarefas a serem realizadas e coordenando os resultados obtidos.

5. Resultados Computacionais

O propósito deste experimento foi o de comparar a performance das estratégias de balanceamento de carga estática (`GRASPIteraçãoParalela`) e dinâmica (`GRASPMestreEscravo`) para a paralelização da meta-heurística GRASP. As duas estratégias foram aplicadas a problemas de decomposição de matrizes no contexto de alocação de tráfego em sistemas de satélites TDMA (Prais e Ribeiro [12]). A medida utilizada para avaliar a performance foi o tempo marcado pelo relógio. Foram utilizados três conjuntos de matrizes de tráfego que se diferenciam por sua dimensão (*n*). Cada um dos três conjuntos de problemas descritos em [1, 12] corresponde a seis matrizes de dimensões iguais a 15, 18 e 21.

Os testes computacionais foram conduzidos numa máquina IBM SP2, um sistema de multiprocessadores de memória distribuída que utiliza a troca de mensagens como esquema de comunicação. O *cluster* do SP2 utilizado usa o sistema operacional AIX 3.2.5 e está configurado com 16 processadores RS6000 mod. 370, cada um com 256 Mbytes de memória RAM, 2 Gbytes de espaço em disco e área para arquivos temporários de 516 Mbytes. Utilizou-se o dispositivo HPS (*High Performance Switch*, 80 Mbytes/s) do SP2 para realizar a troca de mensagens entre os processadores pela rede de comunicação. Uma das características do HPS é sua escalabilidade. Essencialmente, todos os nós da rede estão a igual distância de comunicação, com diversos caminhos para evitar gargalos. O protocolo de comunicação utilizado foi o *us* (*user space mode*) da IBM. Este protocolo é um método rápido de comunicação entre processadores, projetado especialmente para ser usado sob o dispositivo HPS do sistema SP. Um único *job* usando o protocolo *us* pode ser executado em um determinado nó da rede por vez [5]. A comunicação entre processadores foi realizada utilizando chamadas de rotinas da biblioteca MPI [9]. Utilizou-se o *shell script* mpcc v1.2.1.0 com a opção -O2 para otimização do código. O *shell script* mpcc compila programas escritos na linguagem C ao mesmo tempo que liga a biblioteca MPI, inserindo automaticamente as rotinas de inicialização do ambiente paralelo. A versão do MPI é a mpich Mpi-1.0.12, de domínio público. O compilador C utilizado foi o xlC v1.1.2.0. Os testes foram conduzidos em processadores não dedicados e com cargas variadas. As medidas de tempo foram obtidas através da rotina MPI_WTIME da biblioteca MPI e excluem custos de inicialização e finalização. A inicialização consiste da aquisição de processadores, do carregamento do código, da alocação de memória virtual, da leitura de dados de entrada e da criação de estruturas de dados, enquanto a finalização consiste da impressão dos resultados e da liberação dos recursos do sistema para que os processadores possam ser utilizados por outras tarefas.

Nos testes realizados tomou-se o número *K* de iterações GRASP igual a 9000 e o número *bloco* de iterações por tarefa como sendo 500, utilizando-se doze processadores no total. Considera-se como uma amostra de tamanho 60 o conjunto de dez execuções para cada um dos seis problemas com cada uma das três dimensões de matrizes de tráfego, totalizando 180

observações. Jain [7] analisa procedimentos estatísticos para comparar dois sistemas com cargas muito similares. Considera-se o tempo marcado pelo relógio das duas execuções simultâneas do GRASP paralelo, utilizando balanceamento estático (*A*) e balanceamento dinâmico (*B*), como um par de observações. Para cada par computa-se a diferença do tempo e calcula-se um intervalo de confiança para a diferença. Se o intervalo de confiança inclui zero, os desempenhos das duas estratégias não são diferentes no nível de confiança estabelecido. Detalhes dos testes são relatados em Alvim [1]. A Tabela 1 sumariza os resultados para cada amostra. A primeira coluna apresenta a dimensão dos problemas testados, as colunas indicadas por $\overline{A}$ e $\overline{B}$ fornecem os tempos médios em segundos das 60 execuções, respectivamente com balanceamento de carga estático e dinâmico, enquanto a coluna indicada por $(\overline{A-B})/\overline{A}$ mostra a diferença relativa média. A última coluna fornece o intervalo de confiança de 95% para a média das diferenças. Como, para uma mesma dimensão, a amostra é grande (mais de 30 observações), utilizou-se o quantil da distribuição normal no cálculo dos intervalos de confiança.

Observa-se na Tabela 1 que o intervalo de confiança das três amostras não inclui zero. Portanto, pode-se dizer com 95% de confiança que o balanceamento de carga dinâmico é mais eficaz do que o balanceamento de carga estático. Ou seja, para aplicações GRASP paralelas que executam muito trabalho (no caso, seis processadores para executar K = 9000 iterações) deve-se optar por um balanceamento de carga dinâmico. A Figura 2 ilustra os intervalos de confiança e permite comparar visualmente a diferença entre as duas estratégias.

n	$\overline{A}$	$\overline{B}$	$(\overline{A-B})/\overline{A}$	IC (95%)
15	3064	2597	15%	[279, 655]
18	7486	6403	14%	[595, 1573]
21	15829	13843	13%	[813, 3159]

Tabela 1: Síntese dos resultados de 180 execuções.

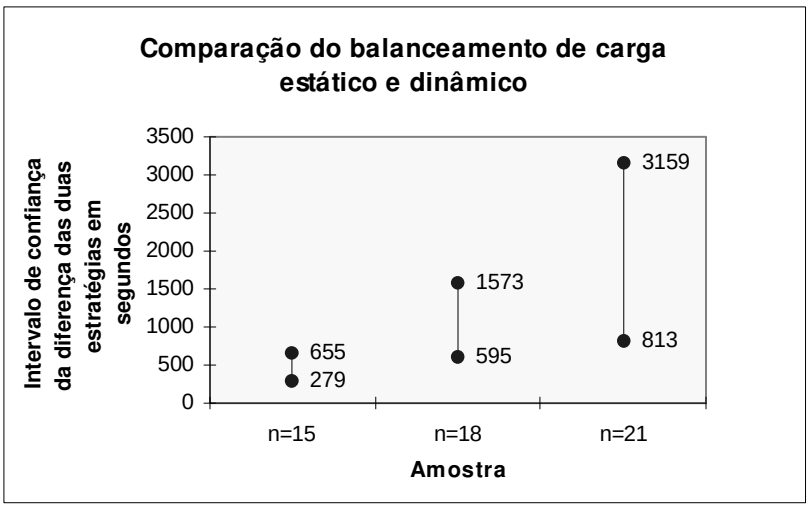

Figura 2: Avaliação do balanceamento de carga do GRASP paralelo.

A eficácia do balanceamento de carga dinâmico foi avaliada levando-se em conta a variabilidade dos dados. Pode-se também observar a razão das médias (Tabela 1) e quantificar em quanto esta estratégia é mais rápida do que o balanceamento de carga estático. Os tempos médios de execução da estratégia que usa balanceamento dinâmico foram de 13 a 15% menores do que os da estratégia com balanceamento estático. Outra interpretação válida considera a probabilidade da estratégia que usa balanceamento dinâmico ser mais rápida. Foram realizadas 180 execuções, das quais em 131 vezes a estratégia que usou balanceamento dinâmico foi a mais rápida. A proporção das vezes em que a estratégia que usou o balanceamento dinâmico foi mais rápida é dada por $q = 131/180 = 0.73$. É possível calcular o intervalo de confiança desta proporção usando o quantil da distribuição normal [7]. O intervalo com 95% de confiança é de [0.66, 0.79]. Como este intervalo não inclui 0.5 (o ponto de igualdade), pode-se dizer com 95% de confiança que o balanceamento dinâmico é mais rápido do que o estático e que em 66 a 79% das execuções a estratégia que usar o balanceamento dinâmico será mais rápida.

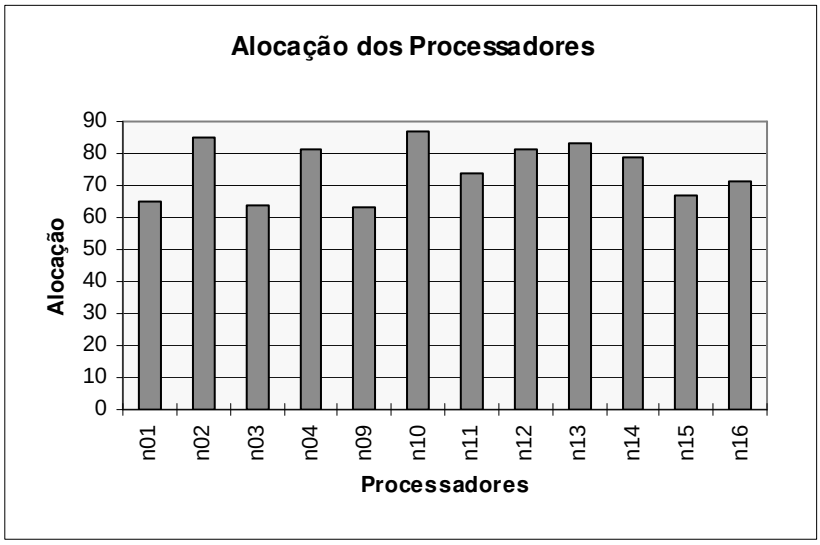

Figura 3: Distribuição da alocação de processadores.

Em relação à estratégia que usa balanceamento dinâmico, algumas análises podem ser feitas a partir dos dados observados. Dos doze processadores disponíveis em cada execução, seis eram alocados aleatoriamente para esta estratégia, dos quais apenas cinco realizavam as iterações GRASP. Logo, nas 180 execuções foram realizadas 900 alocações de processadores para as iterações GRASP. A distribuição das 900 alocações de processadores é mostrada na Figura 3 e permite constatar que a aleatoriedade da alocação de processadores dá oportunidades muito similares para os 12 processadores, mostrando que o mecanismo descrito na Seção 4 não é tendencioso. O processador identificado por n09 foi o menos alocado (63 vezes), enquanto aquele identificado por n10 foi o mais alocado (87 vezes). Uma vez alocado, o processador realiza um número de tarefas que depende da sua carga. Considerando o total de 3240 tarefas, a distribuição da carga de trabalho realizada por cada processador pode ser visualizada através do histograma apresentado na Figura 4, que ilustra a distribuição relativamente uniforme das tarefas entre os processadores disponíveis. O processador com menor carga (n16) executou 223 tarefas e o com maior carga (n13) executou 302 tarefas.

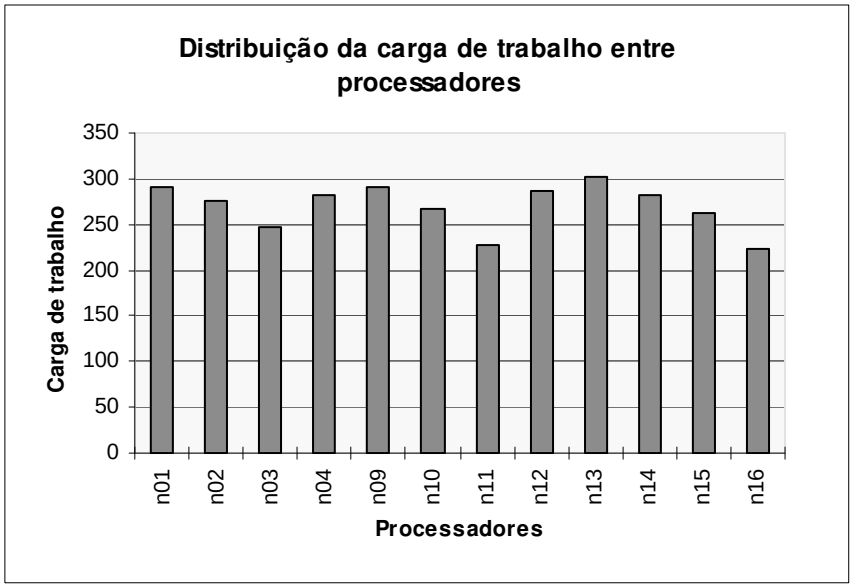

Figura 4: Distribuição da carga de trabalho entre os processadores.

	Freqüência			
Categoria	*n* = 15	*n* = 18	*n* = 21	Acumulado
2	37	41	56	134
3	120	116	83	319
4	93	86	102	281
5	29	41	45	115
6	18	11	12	41
7	3	5	2	10
Total	300	300	300	900

Tabela 2: Freqüência de execução por número de tarefas.

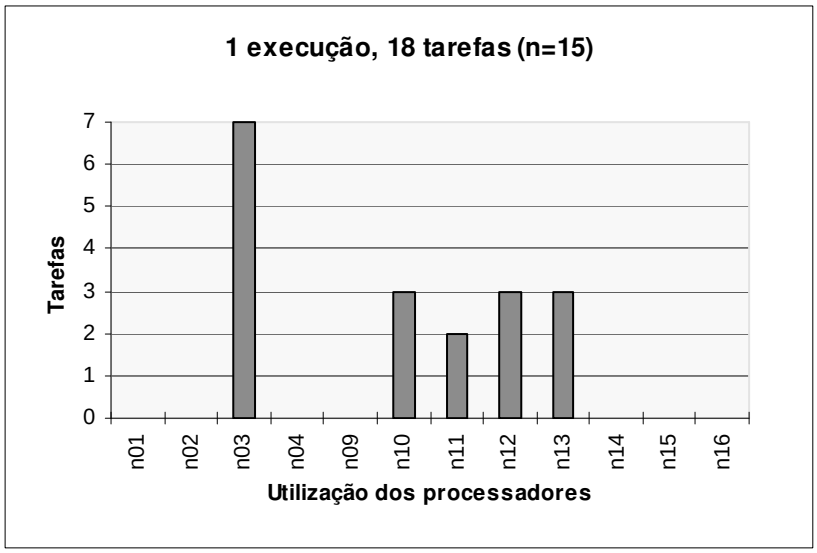

Figura 5: Utilização dos processadores em uma execução.

Outra informação que também pode ser extraída dos dados observados sob balanceamento dinâmico é a distribuição de freqüência do número de tarefas realizadas pelos processadores. Considerando-se cada execução, o número de iterações é igual a 9000 e o número de iterações por bloco é 500. Logo, há 18 tarefas a serem distribuídas por cinco processadores. Como todos os *escravos* recebem uma primeira tarefa, cada um deles realizará de uma a 14 tarefas. Cada uma das três amostras de 60 execuções possui então 1080 tarefas, totalizando 3240 tarefas. O gráfico da Figura 5 ilustra a freqüência de utilização dos processadores, em uma execução de um problema de dimensão n = 15. Dentre os cinco processadores *escravos*, um processador executou sete tarefas, três executaram três tarefas cada um e o outro processador executou apenas duas tarefas (totalizando as 18 tarefas). Para cada execução, é possível classificar as ocorrências em categorias segundo o número de tarefas realizadas por algum processador. A Tabela 2 apresenta estas freqüências totalizadas por amostra (n = 15, 18 e 21) e o resultado acumulado. O histograma da Figura 6 permite visualizar a distribuição dos resultados acumulados. O menor número de tarefas realizadas por algum processador em alguma execução foi igual a dois e o número máximo de tarefas realizadas foi igual a sete. Com freqüência muito grande, um processador executou três ou quatro tarefas.

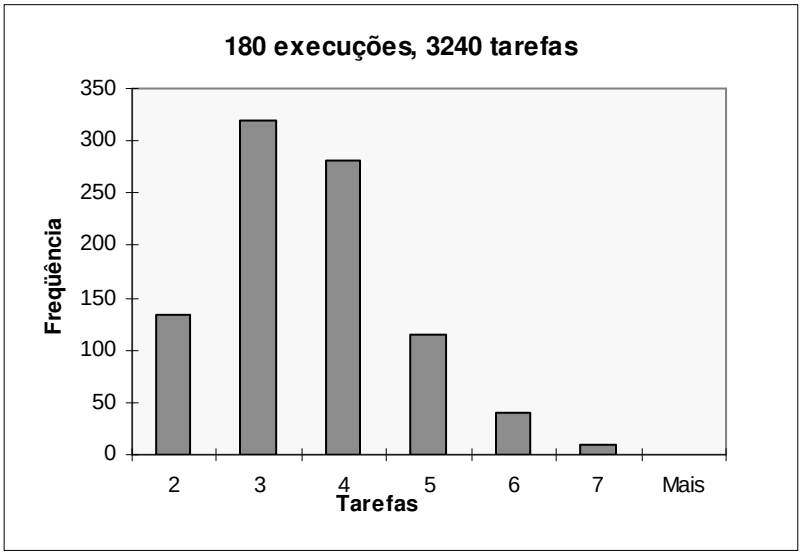

Figura 6: Histograma dos resultados acumulados para as 180 execuções.

6. Conclusões

A meta-heurística GRASP vem sendo aplicada com sucesso na solução de problemas complexos de otimização combinatória [e.g. 6, 12, 13]. Esta caracteriza-se pela independência dos resultados obtidos em cada iteração, o que resulta em implementações paralelas eficientes distribuindo-se as iterações entre os processadores. O custo de comunicação é baixo, visto que uma única variável global é necessária: a melhor solução.

Em um ambiente com processadores não dedicados, a estratégia de paralelização pela distribuição uniforme das iterações (balanceamento de carga estático) pode não ser a melhor

opção em termos de balanceamento de carga. De forma alternativa, pode-se distribuir as iterações entre os processadores sob demanda, conforme sua disponibilidade (balanceamento de carga dinâmico). O desenvolvimento e implementação de um experimento possibilitou que estas estratégias fossem avaliadas com base em execuções simultâneas, usando o mesmo número de processadores diferentes. Um amplo número de testes e execuções permitiu que a comparação do desempenho (medido pelo tempo marcado pelo relógio) das duas estratégias fosse realizada sob a influência dos mesmos fatores externos.

A análise dos resultados computacionais foi feita sobre três amostras de tamanho igual a 60, totalizando 180 observações. Para cada amostra, utilizou-se como medida o intervalo de confiança da diferença dos tempos marcados pelo relógio para as duas estratégias. Em todas as amostras, pode-se afirmar com 95% de confiança que o balanceamento de carga dinâmico é mais eficaz que o estático. Os tempos médios com balanceamento dinâmico foram da ordem de 13 a 15% menores do que com balanceamento estático. Diferentes análises levaram à mesma conclusão: a eficácia do balanceamento de carga dinâmico em relação ao estático. Nota-se ainda que, enquanto o balanceamento estático dispõe de p processadores para realizar as iterações GRASP, o balanceamento dinâmico dispõe de apenas p - 1. No entanto, a sobrecarga de alguns dos p processadores tornou a alternativa de distribuir dinamicamente as iterações uma melhor opção.

As primeiras aplicações paralelas de GRASP surgidas recentemente [10, 11] utilizaram uma distribuição uniforme das iterações entre processadores (balanceamento estático). Os experimentos descritos neste trabalho mostraram que a distribuição dinâmica de iterações é a melhor alternativa para tratar o balanceamento de carga na paralelização do GRASP. Estratégias de paralelização do GRASP baseadas nas conclusões deste trabalho foram aplicadas recentemente com sucesso na solução de problemas de alocação de tráfego em satélites TDMA [1] e de grandes instâncias do problema de Steiner em grafos [8].

Agradecimentos: Os autores agradecem o Laboratório Nacional de Computação Científica (LNCC) pelo uso do sistema IBM SP onde foram realizados os experimentos computacionais.

Referências Bibliográficas

[1] A.C.F. Alvim, *Estratégias de Paralelização da Meta-heurística GRASP*, Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 1998.

[2] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende e W.R. Stewart Jr., "Designing and Reporting on Computational Experiments with Heuristic Methods", *Journal of Heuristics* 1 (1995), 9-32.

[3] R.S. Barr e B.L. Hickman, "On Reporting the Speedup of Parallel Algorithms: A Survey of Issues and Experts", *in* O. Balci, R. Sharda e S.A. Zenios (editores), *Computer Science and Operations Research - New Developments in Their Interfaces*, Pergamon Press, 1992, New York, 279-293.

[4] Cornell Theory Center, "Introduction to Parallel and Distributed Memory Programming", *Index of Training Material by Subject*, 1997.

[5] Cornell Theory Center, "Performance Issues", *Index of Training Material by Subject*, 1997.

[6] T.A. Feo e M.G.C. Resende, “Greedy Randomized Adaptive Search Procedures”, *Journal of Global Optimization* 6 (1995), 109-133.

[7] R. Jain, *The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley, 1991, New York.

[8] S.L. Martins, C.C. Ribeiro e M.C. Souza, “A Parallel GRASP for the Steiner Problem in Graphs”, Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel, Berkeley, 1998, a ser publicado em *Lecture Notes in Computer Science*.

[9] MPI Forum, “MPI: A Message-Passing Interface Standard”, 1995.

[10] P.M. Pardalos, L. Pitsoulis, T. Mavridou, e M.G.C. Resende, “Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP”, *Lecture Notes in Computer Science* 980 (1995), Springer-Verlag, Berlim, 317-331.

[11] P.M. Pardalos, L. Pitsoulis e M.G.C. Resende, “A Parallel GRASP for MAX-SAT Problems”, *Lecture Notes in Computer Science* 1180 (1996), Springer-Verlag, Berlim, 575-585.

[12] M. Prais e C.C. Ribeiro, “Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment”, *INFORMS Journal on Computing*, 1998, a ser publicado.

[13] M.G.C. Resende e C.C. Ribeiro, “A GRASP for Graph Planarization”, *Networks* 29 (1997), 173-189.