Scatter Search Paralelo Aplicado ao Problema do Semáforo

Vitor Andrade dos Santos[1]

[1]Instituto Multidiciplinar – Universidade Federal Rural do Rio de Janeiro (UFRRJ)
Av. Gov. Roberto Silveira – Moquetá, Nova Iguaçu – RJ – 26020-740

[2]Departamento de Ciência da Computação – UFRRJ

vitor@librem.one

Abstract. *The Traffic Light problem is an NP-hard problem which consists on finding a setting for a traffic network which minimizes the total waiting time of the vehicles in the network. In this paper, a parallel heuristic based on the Scatter Search meta-heuristic is proposed for finding good sub-optimal solutions in shared-memory systems in a reasonable time. The Scatter Search is an evolutionary meta-heuristic which utilises more strategic designs over randomization, exploring the search space by means of intensification and diversification of a set of solutions.*
Keywords. *parallel heuristic, scatter search, traffic light, NP-hard*

RESUMO. *O problema do semáforo é um problema NP-difícil que consiste em encontrar uma configuração para uma rede de tráfego que minimize o tempo de espera total dos veículos na rede. Neste artigo, uma heurística paralela baseada na meta-heurística Scatter Search é proposta para encontrar boas soluções sub-ótimas em sistemas de memória compartilhada em um tempo razoável. O Scatter Search é uma meta-heurística que utiliza modelos mais estratégicos ao invés de aleatoriedade, explorando o espaço de busca por meio de intensificação e diversificação de um conjunto de soluções.*
PALAVRAS CHAVE. *heurística paralela, scatter search, semáforo, NP-difícil.*

1. Introdução

O fluxo do tráfego nas cidades é impactado por diversos fatores, dentre eles o tempo de espera dos veículos nos semáforos. Neste trabalho, buscamos um algoritmo capaz de escolher os tempos de início da luz verde de todos os semáforo em uma rede de tráfego de modo a reduzir o tempo total de espera dos veículos. O problema do semáforo foi estudado sob uma variedade de condições, sendo propostos diversos métodos para seu tratamento [1]. Devido à alta complexidade do problema, algumas suposições são definidas de maneira a simplificá-lo:

- Cada semáforo possui apenas duas luzes, verde e vermelha;
- Todos os semáforos possuem um mesmo ciclo T;
- A duração da luz verde é igual a duração da luz vermelha em todos os semáforos;
- Cada carro leva uma unidade de tempo para percorrer uma unidade de distância;
- Quando um veículo chega em uma interseção e a luz do semáforo em sua direção está vermelha, este deve esperar até que a luz fique verde e então seguir até a próxima interseção;

2. Definição do Problema

As instâncias do problema são representadas por um grafo não-direcionado $G = (V, E)$ onde cada vértice $u \in V$ representa uma interseção e cada aresta $(u, v) \in E$ corresponde a um segmento de rua que conecta as interseções u e v. Cada aresta possui um valor associado $d(u, v)$ representando a distância entre as interseções u e v, onde $0 < d(u, v) < T, d(u, v) \in \mathbb{N}$. Um veículo demora uma unidade de tempo para percorrer uma unidade de distância, logo $d(u, v)$ também representa o tempo de trânsito entre u e v. Em uma versão simplificada do problema é considerado que existe somente um semáforo em cada interseção cujo tempo de início da luz verde é indicado por $t(u)$. Já na versão realista é assumido que existe um semáforo para cada aresta (u, v) incidente no vértice u cujo tempo de início de luz verde é indicado por $t(u, v)$. A luz verde se mantém acesa por $T/2$ unidades de tempo, passado esse tempo se torna vermelha e assim permanece por mais $T/2$ unidades de tempo.

Qualquer instância do problema na versão realista pode ser reduzida em tempo polinomial para uma instância equivalente na versão simplificada [1]. Ao tratar o problema é utilizada sempre a versão simplificada da modelagem, devido à sua maior simplicidade. A figura 1 apresenta um exemplo de uma rede de tráfego representada pelo modelo simplificado onde cada vértice possui um valor associado $t(u)$ e cada aresta possui um valor associado $d(u, v)$.

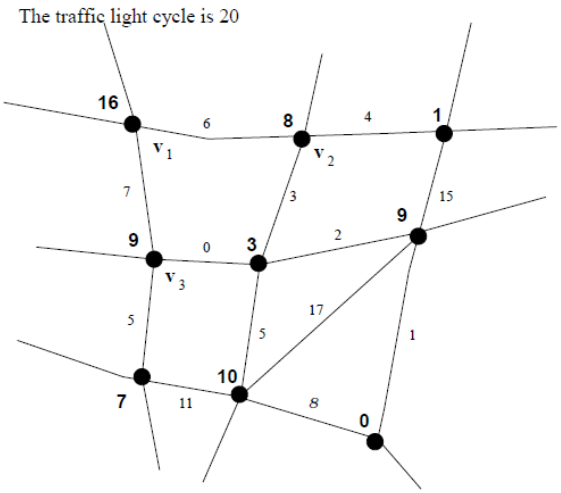

Figura 1. Exemplo de representação de uma rede de tráfego

2.1. Função Objetivo

Para definição da função objetivo é primeiramente definida uma penalidade para um percurso entre dois vértices $u, v|(u, v) \in E$. Dado um percurso de u até v, sua penalidade é definida como o intervalo máximo no qual existe alguma espera ao se transitar de u até v. Considerando que um veículo saia de u em direção à v em um instante i, existirá alguma espera caso $(i + d(u, v)) \bmod T$ pertença ao intervalo no qual a luz do semáforo em v está vermelha. Um veículo pode sair de u somente durante o intervalo no qual o semáforo em u está com a luz verde acessa.

A figura 2(b) exibe um exemplo onde um veículo transita de v_1 para v_2. A linha mais fina representa o tempo em que o semáforo em v_1 está com a luz verde acesa e a linha

mais grossa o intervalo em que o semáforo em v_2 está com a luz verde acessa. Ao sair de v_1 no intervalo $[16, 2)$, o veículo chegará em v_2 no intervalo $[2, 8)$ e o semáforo em v_2 estará com sua luz vermelha, fazendo com que o veículo espere. Portanto, a penalidade de v_1 para v_2 será 6, que é o tamanho do intervalo $[16, 2)$. A penalidade é formalmente definida como:

$C_{uv} = |t(v) - d(u, v) - t(u)| \; mod \; T$ se $(u, v) \in E$
$C_{uv} = 0$ se $(u, v) \notin E$
$P_{uv} = min\{C_{uv}, T - C_{uv}\}$

A função de custo é então definida como $\sum_{u,v \in V} P_{uv}$

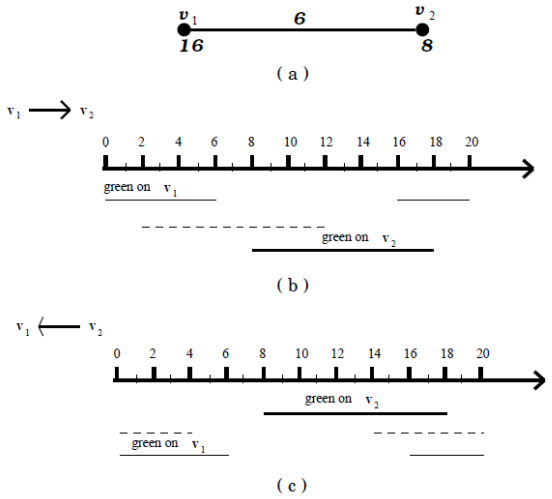

Figura 2. (a) As configurações $t(v_1)$,$t(v_2)$ em uma aresta (v_1, v_2). (b) A penalidade de v1 para v2. (c) A penalidade de v2 para v1

3. Scatter Search

O Scatter Search é uma meta-heurística populacional capaz de encontrar soluções sub-ótimas de boa qualidade para os mais diversos problemas de otimização [3, 4, 5]. Se baseia na ideia de que o espaço de busca pode ser melhor explorado por meio da divisão de uma população em um conjunto elite e um conjunto diverso. O conjunto elite contém soluções de alta qualidade enquanto o conjunto diverso possui soluções de alta diversidade, sendo a união dos dois conjuntos denominada conjunto referência. O template do Scatter Search proposto por Glover [2,6] pode ser sumarizado em quatro métodos:

1. Geração de soluções
2. Aprimoramento de soluções
3. Combinação de soluções
4. Atualização do conjunto referência

Para paralelização da heurística diferentes populações são processdas, uma em cada unidade de processamento. É então definido um conjunto adicional denominado conjunto candidato que contém todas as soluções não incluídas no conjunto referência de uma população por não terem qualidade ou diversidade boas o suficiente. Os conjuntos candidatos são repassados entre as unidades de processamento por meio de um método de

compartilhamento. Durante o compartilhamento a população de cada unidade de processamento pode incorporar soluções dos conjuntos candidatos de outras populações caso essa operação melhore a população em qualidade ou diversidade.

Algorithm 1: Scatter Search

Input: Grafo $G = (E, V)$, tam. da população elite e, tam. da população diversa d
Output: solução S
Inicio:
$U \leftarrow unidades\ de\ processamento\ disponiveis$**;**
for $t \in U$ **do**
  $p_t \leftarrow populacao\ de\ tamanho\ e + d\ criada\ com\ algoritmo\ de\ construcao\ inicial$ **;**
  for $i \in [0, e)$ **do**
    aprimorarMaisIntensivamente(p_{ti}**)**
  for $i \in [e, d)$ **do**
    aprimorar(p_{ti}**)**
  $conjuntoReferencia_t \leftarrow p_t$
while $\neg$ *criterioDeParada()* **do**
  for $t \in U$ **do**
    $subconjunto \leftarrow$ **conjunto de pares distintos do** $conjuntoReferencia_t$**;**
    $conjuntoCandidato \leftarrow \emptyset$**;**
    for $(e1, e2) \in subconjunto$ **do**
      $c \leftarrow$ **combinar(**$G, e1, e2$**);**
      aprimorar(c**);**
      $conjuntoCandidato.adicionar(c)$**;**
    $conjuntoElite_t \leftarrow e$ **melhores solucoes em** $conjuntoReferencia_t \cup conjuntoCandidato$**;**
    diversificar($conjuntoElite_t$**,** $conjuntoReferencia_t$**,** $conjuntoCandidato$**);**
  compartilharConjuntosCandidatos(U**)**
Retornar melhor solucao encontrada;
Fim

3.1. Geração de soluções

Para geração de soluções é primeiramente utilizada uma heurística construtiva gulosa e aleatória para geração de $e + d$ soluções, onde e representa o número de soluções do conjunto elite e d o número de soluções do conjunto candidato. Após geradas as soluções o método de aprimoramento é aplicado nas soluções. Devido ao fato de o Scatter Search possuir melhores resultados quando o conjunto referência inicial possui soluções de boa qualidade e soluções diversas [2] o método de aprimoramento é aplicado de maneira mais intensiva em e soluções.

Algorithm 2: Heurística Construtiva

Input: Grafo G=(V, E)
Output: Vetor de solução
Inicio:
$n \leftarrow quantidade\ de\ vertices\ em\ V$**;**
$solucao \leftarrow vetor\ com\ n\ posicoes$**;**
while *existe vértice não visitado em V* **do**
  $(u, v) \leftarrow aresta\ aleatoria \in E | \neg(visitado_u \wedge visitado_v)$**;**
  $duplas\ candidatas \leftarrow k\ duplas\ aleatorias\ com\ valores\ em\ [0, T)$**;**
  $melhor\ penalidade \leftarrow \infty$**;**
  for each $(t_u, t_v) \in duplas\ candidatas$ **do**
    Atribuir tempo t_u **a** u**;**
    Atribuir tempo t_v **a** v**;**
    $penalidade \leftarrow P_{uv} + P_{vu}$**;**
    if *penalidade* $<$ *melhor penalidade* **then**
      $melhor\ penalidade \leftarrow penalidade$**;**
      $melhor_u \leftarrow t_u$**;**
      $melhor_v \leftarrow t_v$**;**
  $solucao_u \leftarrow melhor_u$**;**
  $solucao_v \leftarrow melhor_v$**;**
  $visitado_u \leftarrow verdadeiro$**;**
  $visitado_v \leftarrow verdadeiro$**;**
Retornar $solucao$**;**
Fim

3.2. Aprimoramento de soluções

Para o aprimoramento das soluções é aplicada uma busca local por Hill Climbing. A cada iteração da busca, é aplicada uma perturbação aleatória em $t(u)$, onde u é um vértice aleatório $\in V$. Caso a perturbação melhore a qualidade da solução então é aceita, caso contrário a perturbação é revertida. A perturbação aplicada afeta somente P_{uv} e P_{vu} $\forall v$ vizinho de u, o que faz com que cada iteração tenha complexidade $O(|N|)$ onde N é o conjunto de vizinhos de u. Apesar de no pior caso $|N| = |V|$, esse caso é irrealista sendo $|N|$ um valor extremamente pequeno em instâncias reais do problema. Um número fixo de iterações é utilizado como critério de parada da busca.

Algorithm 3: Busca Local

Input: Grafo $G = (V, E)$, solução inicial S, numero de iteracoes k
Output: Solução S'
$S' \leftarrow S$;
while $iteracoes < k$ **do**
    $u \leftarrow vertice\ aleatorio\ em\ S'$;
    $melhor_u \leftarrow u\ em\ S'$;
    $N \leftarrow vertices\ vizinhos\ de\ u$;
    $penalidade\ antiga \leftarrow \sum_{v \in N} P_{uv} + P_{vu}$;
    perturbar $t(u)$ em S';
    $penalidade\ nova \leftarrow \sum_{v \in N} P_{uv} + P_{vu}$;
    if $penalidade\ nova > penalidade\ antiga$ **then**
        reverter pertubacao de $t(u)$ em S'
Retornar S';
Fim

3.3. Combinação de Soluções

Para combinação de soluções é proposto um método que explora propriedades específicas do problema de maneira a preservar propriedades das soluções utilizadas na combinação. Dado um grafo $G = (E, V)$, é feita sua divisão em dois subgrafos conexos $G' = (E', V')$ e $G'' = (E'', V'')$ por meio de um corte. Dadas duas soluções $S1$ e $S2$, é criada uma nova solução S onde, para todo vértice $u \in V$:

$S_u = S1_u$ se $u \in V'$

$S_u = S2_u$ se $u \in V''$

Algorithm 4: Combinar

Input: grafo G, solucoes S1, S2
Output: solucao S
Inicio:
Subdividir G **em** G' **e** G''**;**
$S \leftarrow solucao\ vazia$**;**
for $u \in G'$ **do**
    $S_u \leftarrow S1_u$**;**
for $u \in G''$ **do**
    $S_v \leftarrow S2_u$**;**
Retornar S
Fim

3.4. Atualização do Conjunto Referência

O processo de atualização constrói um novo conjunto referência, comparando soluções no conjunto diverso atual com soluções do conjunto candidato e selecionando soluções que aumentem a diversidade do novo conjunto referência.

Determinar um subconjunto de diversidade máxima dentro um conjunto qualquer é um problema NP-Difícil conhecido como o problema da diversidade máxima (Maximum

Diversity Problem, abreviado MD ou MDP). Muitos artigos da literatura recomendam a utilização de uma heurística construtiva gulosa para a diversificação [2], que consiste em sempre escolher a solução cuja distância mínima para todos as soluções já adicionadas ao conjunto referência seja máxima, lembrando que o conjunto referência sempre conterá as e soluções de menor custo independentemente da diversidade dessas soluções. Outras heurísticas podem ser aplicadas para diversificação e existem também métodos exatos para solução do MDP, estes segundos porém possuem custos computacionais proibitivos. A seguir é apresentada a heurística construtiva gulosa de complexidade $O(n^2)$ para diversificação.

Algorithm 5: Diversificação

Input: populacao elite E, populacao diversa D, populacao candidata C
Output: $conjuntoReferencia$
Inicio:
$solucaoEscolhida \leftarrow null$**;**
$conjuntoReferencia \leftarrow E$**;**
$P \leftarrow D \cup C$**;**
$maiorDistanciaMinima \leftarrow -\infty$**;**
for $p \in P$ **do**
    $p.distanciaMinima \leftarrow \infty$**;**
    for $r \in conjuntoReferencia$ **do**
        $d \leftarrow$ **distanciaEntre(**p, r**);**
        if $d < p.distanciaMinima$ **then**
            $p.distanciaMinima \leftarrow d$
    if $p.distanciaMinima > maiorDistanciaMinima$ **then**
        $maiorDistanciaMinima \leftarrow p.distanciaMinima$**;**
        $solucaoEscolhida \leftarrow p$**;**
while $|conjuntoReferencia| < |E| + |D|$ **do**
    $conjuntoReferencia.adicionar(solucaoEscolhida)$**;**
    $P.remover(solucaoEscolhida)$**;**
    $proximaSolucaoEscolhida \leftarrow null$**;**
    $maiorDistanciaMinima \leftarrow -\infty$**;**
    for $p \in P$ **do**
        $distancia \leftarrow$ **distanciaEntre(**$p, solucaoEscolhida$**);**
        if $d < p.distanciaMinima$ **then**
            $p.distanciaMinima \leftarrow d$**;**
        if $p.distanciaMinima > maiorDistanciaMinima$ **then**
            $maiorDistanciaMinima \leftarrow p.distanciaMinima$**;**
            $proximaSolucaoEscolhida \leftarrow p$**;**
    $solucaoEscolhida \leftarrow proximaSolucaoEscolhida$
Retornar $conjuntoReferencia$**;**
Fim

3.5. Compartilhamento de conjuntos candidatos

Para paralelização do Scatter Search são distribuídas uma população para cada unidade de processamento. Os conjuntos candidatos de cada população são comunicados entre algumas unidades de processamento, denominadas unidades mestre, por meio de uma comunicação em árvore. Cada unidade mestre avalia um conjunto candidato e distribui indivíduos entre as populações de suas unidades de processamento escravas conforme os indivíduos demonstram ser benéficos para a qualidade ou diversidade de uma população. Ao final das avaliações um novo conjunto candidato é gerado e comunicado à próxima unidade mestre. Para melhorar a utilização do hardware disponível as unidades mestre podem distribuir tarefas entre as unidades escravas.

Algorithm 6: compartilharConjuntosCandidatos

Input: Conjunto de unidades de processamento U
Output: $conjuntoCandidato$
Inicio:
if $|U| = 1$ **then**
  retornar conjunto candidato em U_0**;**
else
  $U' \leftarrow \{U_i \mid 0 \leq i < |U|/2\}$**;**
  $U'' \leftarrow \{U_i \mid |U|/2 \leq i < |U|\}$**;**
  $C' \leftarrow$ **compartilharConjuntosCandidatos(**U'**);**
  $C'' \leftarrow$ **compartilharConjuntosCandidatos(**U''**);**
  avaliarEIncorporar(U', C''**);**
  avaliarEIncorporar(U'', C'**);**
  retornar $C' \cup C''$**;**
Fim

Algorithm 7: avaliarEIncorporar

Input: Conjunto de unidades de processamento U, Conjunto Candidato C
Inicio:
for $u \in U$ **do**
  $E \leftarrow$ **conjunto elite em** u**;**
  $D \leftarrow$ **conjunto diverso em** u**;**
  for $p \in C$ **do**
    $p.distanciaMinima \leftarrow \infty$**;**
  $melhorCandidato \leftarrow$ **solucao de penalidade minima em** C**;**
  for $e \in E$ **do**
    if $e.penalidade > melhorCandidato.penalidade$ **then**
      trocar($e, melhorCandidato$**);**
      $melhorCandidato \leftarrow$ **solucao de penalidade minima em** C**;**
    recalcularDistancias();
  $jaVerificados \leftarrow \emptyset$**;**
  $melhorCandidato \leftarrow$ **solucao de diversidade minima maxima em** C**;**
  $melhorAtualmente \leftarrow$ **solucao de diversidade minima maxima em** D**;**
  for $i \in [0, |D|)$ **do**
    if $melhorAtualmente.distanciaMinima >$
     $melhorCandidato.distanciaMinima$ **then**
      $jaVerificados.adicionar(melhorAtualmente)$**;**
      $melhorAtualmente \leftarrow$ **solucao de diversidade minima maxima**
       em $D - jaVerificados$**;**
    else
      trocar($melhorAtualmente, melhorCandidato$**);**
      $melhorCandidato \leftarrow$ **solucao de diversidade maxima em** C**;**
    recalcularDistancias();
Fim

4. Experimentos e Resultados

Toda a implementação da heurística foi feita utilizando a linguagem C++ no standard 17, com o compilador GCC versão 7.4.0 e diretiva de otimização *Ofast*. A implementação pode ser encontrada no repositório Git *https://github.com/rockerbacon/traffic-light-search-heuristic*. Para validação da eficácia da heurística foi gerada uma instância aleatória do problema que consiste em um grafo conexo com 500 vértices com graus entre 4 e 10. A heurística é então executada 20 vezes para cada cenário. Os cenários cosistem em configurações com 1, 2, 4, 8 e 16 threads onde as populações sempre totalizam 64 soluções elite e 1280 soluções diversas. O critério de parada do método de intensificação é de 20000 iterações e o critério de parada do Scatter Search varia de acordo com o que se deseja medir. A máquina utilizada nos testes possui a seguinte configuração:

- Processador AMD Ryzen 7 3700X de 8 núcleos com SMT (2 threads por núcleo)
- 16gb de memória RAM DDR4 3000Mhz
- Sistema Operacional Linux Kernel Low-Latency versão 5.0.0

4.1. Speedup

Para analisar o speedup obtido com a paralelização da heurística foi utlizado um número fixo de 25 iterações como critério de parada. A tabela 1 apresenta os tempos médio de execução de cada cenário (em milissegundos) e o speedup obtido com precisão de duas casas decimais. O speedup é calculado pela fórmula t_n/t_1 onde t_n é o tempo médio do cenário com n threads. A variância entre as execuções é desprezível. Os resultados demonstram speedup quase linear para um número de threads ≤ 4. Apesar do speedup piorar consideravelmente com o aumento do número de threads, a implementação apresenta bom aproveitamento do SMT e consegue obter speedup quase linear ao número de núcleos físicos quando o recurso é utilizado.

Threads	tempo médio	speedup
1	65247	1
2	32944	1.98
4	17159	3.8
8	10386	6.28
16	8692	7.51

Tabela 1. Speedup obtido na execução de 25 iterações

4.2. Qualidade das Soluções

Na medição da qualidade das soluções procura-se avaliar o quanto a qualidade das soluções varia em função do número de threads. Foram utilizados dois conjuntos de cenários, um utilizando como critério de parada um tempo fixo de 10 segundos e o segundo um tempo fixo de 30 segundos. Como o custo mínimo para a instância é desconhecido, utilizaremos o trabalho de Yang e Yeh [1] para melhor interpretação dos resultados. O lower bound l é um valor $\leq$ o custo solução ótima. Quanto mais complexa a rede de tráfego mais o custo das soluções tendem a se distanciar de l, enquanto em grafos planares Yang e Yeh conseguiram soluções com custo de aproximadamente $1.18l$ em grafos gerais não foi possível obter uma solução de custo menor que $1.4l$. O lower bound também evidencia o quanto diferenças aparentemente pequenas na função de custo na verdade representam grandes diferenças na qualidade das soluções, visto que soluções aleatórias possuem custo próximo de $2l$. A instância utilizada possui $l = 11642$, sendo consideravelmente mais complexa que as instâncias exploradas por Yang e Yeh, cujo lower bound médio é de 7505.6 para instâncias com a mesma quantidade de vértices. As tabela 2 e 3 apresentam os valores mínimos, médio e máximo da função de custo para os valores encontrados assim como a variância e custo em função do lower bound.

Os resultados evidenciam que a utilização de mais threads resulta em soluções de melhor qualidade em todas as métricas avaliadas. A heurística porém encontra dificuldades para convergir conforme as soluções se aproximam de um ótimo local consideravelmente baixo. Isso é evidenciado pelo aumento na variância ao se aumentar o tempo de execução e a dificuldade em se reduzir o custo máximo em execuções mais longas e com um número maior de threads.

4.3. Qualidade da Paralelizacao

Para testar a qualidade da estratégia de paralelização é utilizado um cenário com um número fixo de 50 iterações e então comparada a qualidade das soluções conforme a

Threads	custo mínimo	custo médio	custo máximo	variância
1	$15570(1.34l)$	$15634.4(1.34l)$	$15694(1.35l)$	38.13
2	$15452(1.33l)$	$15513.7(1.33l)$	$15566(1.34l)$	29
4	$15336(1.32l)$	$15374.2(1.32l)$	$15438(1.33l)$	27.94
8	$15068(1.29l)$	$15127.6(1.3l)$	$15190(1.3l)$	33.27
16	$15028(1.29l)$	$15051.4(1.29l)$	$15088(1.3l)$	16.62

Tabela 2. Qualidade das soluções obtidas em 10 segundos

Threads	custo mínimo	custo médio	custo máximo	variância
1	$15472(1.33l)$	$15569.4(1.34l)$	$15622(1.34l)$	37.36
2	$15140(1.3l)$	$15212.9(1.31l)$	$15300(1.31l)$	43.44
4	$14976(1.29l)$	$15031.2(1.29l)$	$15100(1.3l)$	36.64
8	$14942(1.28l)$	$14982.2(1.29l)$	$15052(1.29l)$	30.24
16	$14938(1.28l)$	$14975.7(1.28l)$	$15028(1.29l)$	23.65

Tabela 3. Qualidade das soluções obtidas em 30 segundos

quantidade de threads varia. Os resultados apresentados na tabela 4 demonstram não só que a paralelização não impactou negativamente a heurística mas ainda que a estratégia é benéfica para sua convergência, sendo possível que sua utilização possa melhorar o Scatter Search tradicional mesmo em sistemas que não possuam mais de um núcleo de processamento disponível.

Threads	custo mínimo	custo médio	custo máximo
1	15322	15407.8	15482
2	14962	14998.8	15042
4	14968	15010.5	15076
8	14946	15000	15052
16	14958	14992.6	15034
32	14958	14979.7	15002

Tabela 4. Qualidade das soluções obtidas em 50 iterações

5. Conclusão

O trabalho realizado demonstra que o Scatter Search é uma boa alternativa para tratamento do problema do semáforo, sendo possível obter soluções de boa qualidade com baixo risco de altas variações e de maneira extremamente rápida e escalável. Também é demonstrado que é possível paralelizar o Scatter Search de maneira muito eficiente em sistemas de memória compartilhada.

Apesar de o autor considerar os resultados muito satisfatórios algumas lacunas referentes ao problema e à heurística permanecem:

- O indicativo de que o método de distribuição e comunicação entre as populações possa ser benéfico para ambientes single-threaded não foi validado;
- É possível que a dificuldade em melhorar as soluções além de certo ponto ocorra pelas estratégias extremamente simples de busca local e exploração de vizinhança.

Experimentos com outros métodos de combinação, busca local e vizinhança podem ajudar a melhorar a heurística ainda mais;

- Ainda que instâncias do problema realista sejam redutíveis para instâncias do problema simplificado a modelagem do problema realista em si sofre de limitações para aplicações em problemas reais, como a falta de restrições aos tempos de semáforos em uma mesma interseção e a representação do problema de maneira estática;
- Existem poucos trabalhos que tratam do problema do semáforo estático, o que dificulta a comparação da heurística proposta com outras estratégias. A aplicação da mesma estratégia em problemas mais clássicos pode ajudar na validação da heurística

Referências

[1] Yang CB, Yeh YJ. *The Model and Properties of the Traffic Light Problem*. In: Proceedings of international conference on algorithms (1996). Kaohsiung, Taiwan, pp 19–26.

[2] Laguna M, Martí R. *Scatter Search*. In: Metaheuristic Procedures for Training Neutral Networks (2006). University of Colorado, Boulder, Colorado, pp 139-152.

[3] Laguna M. *Scatter Search*. In: Handbook of Applied Optimization (2002). Oxford University Press, New York, pp 183-193.

[4] Laguna M, Martí R. *Scatter Search: Methodology and Implementations in C*. In: ISBN 1-4020-7376-3 (2003). Boston, New York, pp 312.

[5] Laguna M, Martí R. *Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions*. In: Journal of Global Optimization, vol. 33 (2005). pp 235-255

[6] Glover F, Laguna M, Martí R *Fundamentals of Scatter Search and Path Relinking*. In: Control and Cybernetics, Volume 29, Number 3 (2000). pp 653-684