Ilustração do método Generalized DVND

Rodolfo, Igor, Vitor, Nenad

November 1, 2017

Abstract

Your abstract.

1 Ilustração do método DVND

Seja S o espaço de soluções para um problema de otimização Π, e $s \in S$ uma solução qualquer. Uma função objetivo $f : S \to \mathbb{R}$ atribui um valor para cada solução. Denotamos por $N^x(s)$ o conjunto de soluções vizinhas a s (considerando a vizinhança x), usualmente apresentando também operadores $m^x : S \to S$ chamados *movimentos* que levam uma solução s a cada solução vizinha $s' \in N^x(s)$, onde $s' = m^x(s)$. Como s e s' podem ser vistas como funções constantes, podemos escrever $s' = m^x \circ s$.

Para a apresentação do algoritmo proposto, será útil definir um *custo* para cada movimento, representando o impacto deste movimento na solução corrente. Definimos o custo $\widehat{m^x_f}(s)$ com respeito à função f e vizinhança x como $\widehat{m^x_f}(s) = f(s') - f(s)$, onde $s' = m^x \circ s$. Para simplificar a notação, iremos omitir a função f e vizinhança x quando estiverem claras pelo contexto. Formalmente, a notação de chapéu representa uma função $\widehat{m} : S \to \mathbb{R}$.

Para o desenvolvimento do algoritmo, estamos interessados em movimentos com uma característica especial. Denotamos movimentos m_1 e m_2 como *independentes entre si* quando, para uma solução s qualquer, $\widehat{m_1}(s) + \widehat{m_2}(s) = \widehat{m_2}(m_1 \circ s) = \widehat{m_1}(m_2 \circ s)$.

A mesma ideia pode ser aplicada para um conjunto de movimentos $M = \{m_1, m_2, m_3, ...\}$, ditos independentes se para uma solução s qualquer e para todo subconjunto não-vazio $M' = \{m_1, m_2, m_3, ..., m_k\} \subseteq M$ temos $\widehat{m_1}(s) + \widehat{m_2}(s) + \widehat{m_3}(s) + ... + \widehat{m_k}(s) = \widehat{m_1}(m_2 \circ m_3 \circ ... \circ m_k \circ s)$. Utilizaremos a letra I para denotar movimentos independentes, por exemplo, $I(\{m_3, m_5, m_9\})$.

Para detectar quais movimentos são independentes, uma estrutura de dados C para gerenciamento de conflitos foi proposta. No caso do problema em questão, a estrutura se trata de vetor com N bits, onde cada bit i indica se houve alguma alteração relativa ao cliente i.

Exemplo C: |0|0|0|1|1|1|1|0|0|0|0|0|0|

Assim, a estrutura C começa com todos bits zero, considerando uma solução de referência s. Cada movimento altera a estrutura C além de alterar a solução corrente, indicando quais posições estão comprometidas. Da mesma maneira, uma função d de detecção de conflitos indica se um movimento é "aplicável" dada certa configuração da estrutura de conflitos.

No DVND, várias buscas começam de uma mesma solução de referência e o melhor movimento (ou primeiro de melhora) é retornado como candidato à inclusão em um pool de movimentos, chamado Histórico. Este pool pode ser visto como um grafo não direcionado, no qual os vértices são movimentos e arestas indicam conflitos entre movimentos. Porém, setas especiais são utilizadas para marcar dependência entre movimentos. Se m_B depende de m_A, então m_B só pode ser aplicado se m_A for aplicado antes na solução. Escolhendo o maior (ou menor) conjunto independente neste grafo, respeitando os conflitos e dependências, resultará na melhor combinação de movimentos em um dado momento, considerando múltiplas estruturas de vizinhança simultaneamente.

A Figura 1 apresenta um exemplo do grafo de Histórico para uma execução do DVND com três processos de busca.

Para entender melhor a ideia do algoritmo, a tabela abaixo simula a execução do algoritmo para três processos.

O Histórico começa distribuindo a solução s_1 para cada busca, e eventualmente coleta um novo movimento e distribui uma nova solução. Todos os cálculos são feitos em cima de uma mesma solução de referência s_1, que só é modificada no passo 20 do algoritmo. Nas iterações 7 e 9, note que existem conflitos a serem resolvidos (vide grafo). Após a iteração 19, o Histórico percebe

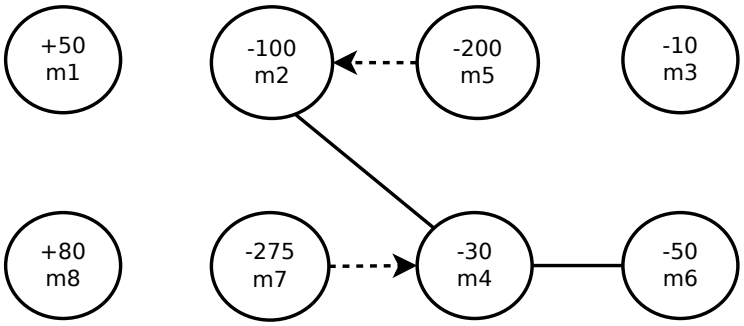

Figure 1: Exemplo de um grafo de Histórico. A combinação ótima de custos consiste nos movimentos m_2, m_3, m_5 e m_6

Table 1: Execução do DVND

#	f	solução	direção	processo
1	1000	s_1	$\rightarrow$	P1
2	1000	s_1	$\rightarrow$	P2
3	1000	s_1	$\rightarrow$	P3
4	970	$m_4 \circ s_1$	$\leftarrow$	P3
5	970	$m_4 \circ s_1$	$\rightarrow$	P3
6	900	$m_2 \circ s_1$	$\leftarrow$	P1
7*	900	$m_2 \circ s_1$	$\rightarrow$	P1
8	1050	$m_1 \circ s_1$	$\leftarrow$	P2
9*	900	$m_2 \circ s_1$	$\rightarrow$	P2
10	890	$m_3 \circ m_2 \circ s_1$	$\leftarrow$	P1
11	890	$m_3 \circ m_2 \circ s_1$	$\rightarrow$	P1
12	700	$m_5 \circ m_2 \circ s_1$	$\leftarrow$	P2
13	690	$m_5 \circ m_3 \circ m_2 \circ s_1$	$\rightarrow$	P2
14	695	$m_7 \circ m_4 \circ s_1$	$\leftarrow$	P3
15	685	$m_4 \circ m_4 \circ m_3 \circ s_1$	$\rightarrow$	P3
16	840	$m_6 \circ m_3 \circ m_2 \circ s_1$	$\leftarrow$	P1
17	640	$m_6 \circ m_3 \circ m_5 \circ m_2 \circ s_1$	$\rightarrow$	P1
18	765	$m_8 \circ m_7 \circ m_4 \circ m_3 \circ s_1$	$\leftarrow$	P3
19	640	$m_6 \circ m_3 \circ m_5 \circ m_2 \circ s_1$	$\rightarrow$	P3

que os três processos tem movimentos em comum: m_3, m_5 e m_2. Então uma nova solução s_2 é armazenada no Histórico como solução de referência, sendo ela $s_2 = m_3 \circ m_5 \circ m_2 \circ s_1$ (de custo 690). Assim, os processos P1 e P3 continuam naturalmente processando a solução $m_6 \circ s_2$ (antiga $m_6 \circ m_3 \circ m_5 \circ m_2 \circ s_1$), e P2 continua com s_2 (antiga $m_5 \circ m_3 \circ m_2 \circ s_1$). Note também que, ao adotar estes três movimentos, todos movimentos com conflitos (ou dependência de algum movimento em conflito) tem de ser eliminados. Então, m_4 e m_7 são descartados. Vale observar que m_1 e m_8 nunca foram armazenados, pois eram de piora. O grafo final então consiste somente dos movimentos m_6, m_3, m_5 e m_2.

Table 2: Consolidação de uma nova solução no DVND

#	f	solução	local
20	690	$s_2 = m_3 \circ m_5 \circ m_2 \circ s_1$	Histórico
	640	$m_6 \circ s_2$	P1
	690	s_2	P2
	640	$m_6 \circ s_2$	P3

Utilizar a CPU para decidir qual SOLUCAO cada processo de busca local irá utilizar, minimizando as transferências de memória CPU-GPU e, assim, maximizando o aproveitamento das ADS na memória da GPU.

EXEMPLO:
Processo A => CPU
Processo B => Kernel GPU Swap
Processo C => Kernel GPU 2-Opt
Processo D => Kernel GPU Or-1
Processo E => Kernel GPU Or-2
Processo F => Kernel GPU Or-3

Solução S1(custo 1000) = [1,2,3,4,5,6,7,8,9,10]

A solução S1+ADS entra na GPU para todas as buscas B-F, cada uma em um fluxo (thread) distinto. Supomos que o processo B (swap) termina antes com ganho -100 (melhora) na troca M1=(2,5). A CPU então cria S2 = S1 ++ M1. Como S2 e a melhor solução atual, ela é copiada para a GPU, e o processo B roda em cima de S2. S2 (custo 900)=[1,5,3,4,2,6,7,8,9,10]

Neste momento, o processo D termina com Or-1 M2=(7,9), movendo cliente 7 para depois do 9), também com ganho -50. A CPU então analisa M1 e M2. Como não há conflitos entre M1 e M2, uma solução S3 = S1 ++ M1 ++ M2 é criada com seu ADS para o processo D executar. S3 (custo 850)=[1,5,3,4,2,6,8,9,7,10]

O processo C termina com o movimento 2Opt M3=(1,3) (inverte de 1 a 3) com ganho -120. A CPU percebe que M3 conflita com M1, mas não com M2. Então a solução S4 (custo 830) = S1 ++ M2 ++ M4 é gerada e passada com ADS à GPU para o processo C.

O processo E termina SEM GANHO com o movimento M4 = (1,4) (move clientes 1,2 para depois do 4). A CPU percebe que M4 conflita com M1 e M3, mas não M2. Porém, a melhor configuração atual é S4 (custo 830), então a busca parte dela (e ela já está na memória da GPU).

Finalmente, o processo B (swap) termina denovo com melhora -60 e troca M5=(3,8). Esta troca roda em cima de S2, então não existem outros movimentos para haver conflito! S5 (custo 840) = S2 ++ M5 é criada com ADS. Porém, a solução S4 (custo 830) é melhor. A CPU então relança o processo B com a solução S4.

Dúvidas (com provável resposta experimental): O tempo gasto pela CPU para escolher a solução vai atrasar muito os processos parados das GPUs? Igor: acho que não, pois serão poucos movimentos candidatos não conflitantes e alguns serão descartados com o tempo.

Qual o critério para descartar movimentos e CONSOLIDAR alguma solução como de referência?

Igor: eu não sei!! Dependendo se for rápido, podemos levar todo o processo com todos os movimentos DE MELHORA, e opcionalmente ir CONSOLIDANDO soluções (utilizá-las como referência-base também) quando X% das posições forem alteradas (próximo de 100% indica que muitos conflitos virão em cima dela).

Igor 2: pensei novamente sobre isso e cheguei a uma possível solução. Digamos que as buscas B-F estão trabalhando nas soluções S1 ++ M1 ++ M2 ++ ..., S1 ++ M2 ++ M4 ++ ... , S1 ++ M2 ++ M8 ++ ..., percebemos que todas incorporaram o M2. Neste caso, a nova solução de referência deve ser X1 = S1 ++ M2. A partir daí, como M2 já é naturalmente incluído, mais opções

de movimentos novos não conflitantes vão surgir e poder competir pelas soluções na memória. Mas qual a frequencia desta convergência? Eu não sei, acredito que seja alta e que funcione. Espero :)

Qual o algoritmo por trás do comportamento da CPU? Ele é exato? Se for exato pesado, pode então ser heurístico? Igor: Eu acho que é encontrar um conjunto independente com custo máximo, onde vértices são movimentos e arestas indicam conflito entre os movimentos. Me parece ser rápido de resolver exato, mas não conheço algoritmos para isso.

Em quais situações o DVND vai se comportar como o VND clássico? Igor: eu não sei. Eu ACREDITO que em geral o tempo vai ser um pouco maior que o VND clássico (mesmo com processamento paralelo), mas a qualidade vai ser bastante superior.

É possível restringir o DVND para ser bem mais rápido que o VND e se comportar "próximo" de um VND? Igor: eu não sei.

Sobre o último caso do algoritmo (entre S5 e S4), a solução melhorada S5 é desperdiçada, mas poderia ser utilizada como escape de um possível ótimo local em S4... Mas se o critério não for a melhor solução corrente, cada busca vai divergir em um caminho distinto, e não vai acontecer mais colaboração entre elas. Será possível encontrar um caminho intermediário entre centralização e diversificação? Igor: Provavelmente é melhor explorar a solução centralizada primeiro. Uma possível alternativa seria restringir uma piora aceitável a X% da melhor solução corrente, mas acho que isso viria como melhoria em cima do trabalho, e o VND viraria uma metaheurística.

1.1 Tabela de conflitos

	$\boldsymbol{swap(i_2, j_2)}$	$\boldsymbol{2\text{-}opt(i_2, j_2)}$	$\boldsymbol{oropt\text{-}k_2(i_2, j_2)}$
$\boldsymbol{swap}$ $\boldsymbol{(i_1, j_1)}$	$(\lvert i_1 - i_2\rvert > 1) \wedge$ $(\lvert i_1 - j_2\rvert > 1) \wedge$ $(\lvert j_1 - i_2\rvert > 1) \wedge$ $(\lvert j_1 - j_2\rvert > 1)$	$[(i_1 < i_2 - 1) \vee$ $(i_1 > j_2 - 1)] \wedge$ $[(j_1 < i_2 - 1) \vee$ $(j_1 > j_2 + 1)]$	$(j_1 < \min(i_2, j_2) - 1) \ \vee$ $(i_1 > \max(i_2, j_2) + k_2) \vee$ $[(i_1 < \min(i_2, j_2) - 1) \ \wedge$ $(j_1 > \max(i_2, j_2) + k_2)]$
$\boldsymbol{2\text{-}opt}$ $\boldsymbol{(i_1, j_1)}$	$[(i_2 < i_1 - 1) \vee$ $(i_2 > j_1 + 1)] \wedge$ $[(j_2 < i_1 - 1)] \vee$ $(j_2 > j_1 + 1)]$	$(j_1 < i_2 - 1) \vee$ $(i_1 > j_2 + 1) \vee$ $(j_2 > i_1 - 1) \vee$ $(i_2 > j_1 + 1)$	$(i_1 > \max(i_2, j_2) + k_2) \vee$ $(j_1 < \min(i_2, j_2) - 1)$
$\boldsymbol{oropt\text{-}k_1}$ $\boldsymbol{(i_1, j_1)}$	$(j_2 < \min(i_1, i_2) - 1) \ \vee$ $(i_2 > \max(i_1, i_2) + k_1) \vee$ $[(i_2 < \min(i_1, i_2) - 1) \ \wedge$ $(j_2 > \max(i_1, i_2) + k_1)]$	$(j_2 < \min(i_1, j_1) - 1) \vee$ $(i_2 > \max(i_1, j_1) + k_1)$	$[\max(i_1, j_1) + k_1 < \min(i_2, j_2)] \vee$ $[\min(i_1, j_1) > \max(i_2, j_2) + k_2]$

Table 3: Tabela de movimentos não conflitantes

References