Otimização de Rotas Logísticas por meio da abordagem *Team Orienteering Problem* baseada em metaheurística GRASP e *Path Relinking*

1st Tiago Funk, 2nd Adriano Fiorese, 3rd Fabiano Baldo
Departamento de Ciência da Computação - DCC
Programa de Pós-Graduação em Computação Aplicada - PPGCAP
Universidade do Estado de Santa Catarina - UDESC
Joinville, SC, Brasil
tiagoff.tf@gmail.com, adriano.fiorese@udesc.br, fabiano.baldo@udesc.br

***Resumo*—Na logística um dos maiores desafios é a elaboração das rotas dos veículos que minimize os deslocamentos e seja realizada em tempo aceitável. Uma abordagem para a otimização do roteamento é a baseada na solução do problema *Team Orienteering Problem* (TOP). Entretanto, assim como o *Traveling Salesman Problem* (TSP), o TOP também é um problema de complexidade computacional *NP-Hard*. Portanto, deve ser resolvido com o auxílio de abordagens metaheurísticas. Nesse sentido, este trabalho apresenta uma abordagem de solução do *Team Orienteering Problem* enquadrada para problemas de otimização de rotas logísticas. Tal abordagem visa o desenvolvimento de um algoritmo baseado na metaheurística GRASP conjugada com a técnica de intensificação de resultados *Path Relinking*. Os resultados alcançados demonstram que o algoritmo proposto alcançou os resultados encontrados na literatura para 46,5% das instâncias executadas, indicando resultados promissores para um trabalho em desenvolvimento.**

I. Introdução

Serviços de compras e agendamento cada vez mais comuns na forma online têm evidenciado a importância da solução de problemas de logística, tanto para a entrega segura de bens como na execução de tarefas no prazo adequado. No caso das entregas, a utilização de um algoritmo que elabore rotas para vários veículos que precisam realizar tal tarefa em conjunto, com um tempo aceitável para a construção da rota, pode ser um diferencial competitivo para empresas. É o caso, por exemplo, de empresas de logística ou agências de turismo que desejam otimizar rotas para maximizar seu lucro e minimizar o custo de deslocamento pela rota.

Portanto, a elaboração dessas rotas deve ser otimizada de forma a alcançar o objetivo esperado. Uma abordagem adequada a otimização dos resultados da execução do roteamento é a baseada na solução do problema *Team Orienteering Problem* (TOP). Ela consiste na solução do problema de execução de tarefas que devem ser realizadas em localizações geográficas distintas, com restrição de tempo, por uma equipe de executores cujo custo total do percurso é pré-determinado. Ainda, devido às restrições de tempo e/ou custo da rota, torna-se inviável a realização de todas as tarefas. Dessa forma, surge a necessidade de escolha das tarefas a serem executadas. Essa escolha precisa respeitar uma ordem de prioridade das tarefas ou de recompensa para a equipe. A solução de escolha das tarefas deve lidar também com a alocação dos indivíduos da equipe de forma a minimizar a alocações deles [1].

Portanto, dentre as abordagens para a solução do problema de roteamento existentes, a abordagem *Team Orienteering Problem* (TOP) foca na maximização do ganho, com restrição do tempo máximo da rota, o que a torna uma estratégia bastante compatível a aplicação na minimização do *Traveling Salesman Problem* (TSP) para problemas de otimização de rotas logísticas.

O *Team Orienteering Problem* é um problema de complexidade computacional *NP-Hard*, pois este é a versão que pode trabalhar com mais de uma rota ao mesmo tempo do *Orienteering Problem*(OP), que já é *NP-Hard* [1]. Portanto, para sua solução de forma robusta é necessário a aplicação de heurísticas ou metaheurísticas que proveem uma solução aproximada, chamada de melhor solução possível. Existem várias abordagens metaheurísticas para a solução de problemas *NP-Hard*. Basicamente, elas podem ser divididas entre aquelas baseadas no melhoramento de solução única ou de uma população. As metaheurísticas populacionais, apesar de serem robustas, consomem muitos recursos computacionais para realizar a otimização de várias soluções ao mesmo tempo. Por outro lado, as metaheurísticas de melhoramento de solução única, além de alcançarem resultados equiparáveis as populacionais, necessitam de menos recursos computacionais para a sua execução.

Dentre as metaheurísticas de melhoramento de solução única se destaca a abordagem *Greedy Randomized Adaptive Search Procedure* (GRASP). O GRASP utiliza uma abordagem iterativa de criação de soluções iniciais gulosas randomizadas, onde são aplicados melhoramentos locais para intensificar o encontro de uma solução ótima local. Esses melhoramentos locais podem ser realizados com a aplicação de operadores tais como, 2-opt, *Path Relinking*, entre outros.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Ao final do ciclo iterativo, o melhor ótimo local encontrado após as n execuções do ciclo é considerado a melhor solução.

Portanto, este trabalho apresenta uma abordagem de solução do TOP enquadrada para problemas de otimização de rotas logísticas. Tal abordagem visa o desenvolvimento de um algoritmo baseado na metaheurística GRASP conjugada com a técnica de intensificação de resultados *Path Relinking*.

Este trabalho está organizado da seguinte forma. A Seção II apresenta sua fundamentação teórica. A Seção III apresenta a revisão de trabalhos relacionados. A seção IV detalha a proposta de solução. A Seção V avalia os resultados obtidos. Por fim, a Seção VI apresenta a conclusão e trabalhos futuros.

II. Fundamentação Teórica

Esta seção apresenta a base teórica necessária à compreensão do trabalho proposto. Portanto, ela define e contextualiza o problema TOP, formaliza a abordagem de solução metaheurística GRASP, bem como os conceitos de *Path Relinking* que serão utilizados de forma a melhorar os resultados produzidos pela metaheurística utilizada.

A. Team Orienteering Problem

O *Team Orienteering Problem* foi mencionado pela primeira vez por [2] como Multiple Tour Maximum Collection Problem (MTMCP). Ele foi introduzido formalmente em [1] como solução para o problema de alocação de membros de uma equipe esportiva. O TOP é uma variante do *Traveling Salesman Problem* (TSP), sendo que o TSP é uma subclasse do problema de *Vehicle Routing Problem* (VRP) [3].

O problema descrito pelo TOP consiste em construir várias rotas para membros de uma equipe. Cada local pelo qual passa uma rota possui uma pontuação que é coletada ao ser visitada. O conjunto de rotas visitadas deve atingir o máximo de pontuação possível para a equipe. A pontuação pode ser definida como lucro, pontos em uma competição esportiva, etc. Um local não pode ser visitado duas vezes, nem mesmo por outro membro da equipe, e existe uma restrição de tempo máximo para cumprir completamente a rota [1].

A descrição formal do problema consiste em um conjunto V de vértices a serem visitados e um conjunto E de arestas entre os vértices em V. Assim $G = V, E$ é um grafo completo. Cada vértice i pertencente a V possui uma pontuação s_i que não pode ser negativa. Cada aresta em E é simétrica, com custo não negativo c_{ij} associado. O custo de uma aresta pode ser a distância entre os vértices ou o custo da viagem. Para resolver o problema devemos encontrar caminhos para todos os M membros da equipe. Todos os caminhos iniciam no vértice v_0 e finalizam no vértice v_n. O vértice v_0 e o vértice v_n podem ser diferentes ou o mesmo dependendo da modelagem. Cada pontuação só é coletada uma vez por um membro da equipe, logo, dois caminhos diferentes não podem passar pelo mesmo vértice. O tempo gasto por cada um dos membros não pode ultrapassar o tempo limite T_{max}.

Este problema é *NP-hard* [1], o que implica que métodos exatos de resolução podem utilizar tempo computacional alto. Outros métodos, como metaheurísticas acabam sendo mais oportunas, pois utilizam menos tempo, embora não garantam a solução ótima. O problema TOP é considerado *NP-hard*, pois a descoberta de tais rotas envolve a combinação de todos os vértices possíveis. Assim, o tempo necessário para se encontrar a resposta ótima cresce exponencialmente conforme cresce a cardinalidade de G.

O TOP foi originalmente modelado como um problema de programação linear em [3]. Porém, naquele modelo o vértice inicial e final eram os mesmos. Em [4], uma modelagem foi derivada para suportar que o vértice inicial possa ser diferente do vértice final. Para a formulação do TOP é necessário definir uma notação adicional:

- $y_{ik} (i = 1, ..., n; k = 1, ..., m)$: 1, se o vértice i é visitado na rota k; 0, caso contrário.
- $x_{ijk} (i, j = 1, ..., n; k = 1, ..., m)$: 1, se a aresta (i, j) é visitada na rota k; 0, caso contrário.
- d_{ij}: distância representada pela aresta $(i, j) \in E$
- Como $c_{ij} = c_{j,i}$, apenas $x_{ijk} (i < j)$ esta definida.
- U: um conjunto de vértices em V, $U \subset V$

Assim, o TOP pode ser representado como o seguinte problema de programação linear:

$$\text{Maximize:} \sum_{i=2}^{n-1} \sum_{k=1}^{m} r_i y_{ik} \tag{1}$$

$$\text{Sujeito a:} \sum_{j=2}^{n} \sum_{k=1}^{m} x_{1jk} = \sum_{i=1}^{n-1} \sum_{k=1}^{m} x_{ink} = m \tag{2}$$

$$\sum_{i<J} x_{ijk} + \sum_{i>j} x_{jik} = 2y_{jk}, (j = 2, ..., n-1; k = 1, ..., m) \tag{3}$$

$$\sum_{k=1}^{m} y_{ik} \leqslant 1, (i = 2, ..., n-1) \tag{4}$$

$$\sum_{i=1}^{n-1} \sum_{j>i} c_{ij} x_{ijk} \leqslant T_{max}, (k = 1, ..., m) \tag{5}$$

$$\sum_{i,j \in U, i<j} x_{ijk} \leqslant |U| - 1, (U \subset V \backslash \{1, n\}; 2 \leqslant |U| \leqslant n-2; k = 1, ..., m) \tag{6}$$

$$x_{ijk} \in \{0, 1\}, (1 \leqslant i < j \leqslant n; k = 1, ..., m) \tag{7}$$

$$y_{1k} = y_{nk} = 1, y_{ik} \in 0, 1 (i = 2, ..., n-1; k = 1, ..., m) \tag{8}$$

A Eq. 1 apresenta a função objetivo que visa maximizar a recompensa total da equipe ao visitar as rotas escolhidas, onde r_i representa o valor de pontuação dos vértices e y_{ik} se este foi visitado. A restrição apresentada pela Eq. 2 indica que cada rota deve iniciar no vértice 1 e finalizar no vértice n, e a quantidade de vezes que os vértices 1 e n são visitados devem ser igual ao número de rotas (m). A restrição apresentada pela Eq. 3 garante a conectividade de cada caminho. Já a restrição apresentada pela Eq. 4 certifica que cada vértice (exceto o 1 e o n) deve ser visitado no máximo uma vez. A restrição 5 descreve a restrição de tempo. A restrição 6 certifica que a existência de sub-caminhos sejam proibida. As restrições 7 e 8 garantem a integridade de cada variável.

A modelagem de programação linear para este problema demonstra que é possível resolvê-lo de forma exata, mas com um custo computacional alto. Assim, metaheurísticas se tornam úteis para diminuir o custo computacional, porém, sem a garantia de soluções ótimas. Metaheurísticas apresentam um conjunto de algoritmos genéricos que podem ser implementados para diversos problemas de otimização, inclusive para o TOP. Neste trabalho será utilizado a metaheurística GRASP.

B. Metaheurísticas

As metaheurísticas são um conjunto de técnicas de aproximação para encontrar soluções em problema de otimização. Essa família de técnicas fornece soluções aceitáveis em um tempo razoável para problemas difíceis e complexos. Mas ao contrário de métodos de otimização exatos, não há garantia que soluções ótimas possam ser obtidas. Metaheurísticas utilizam-se de soluções que são construídas de forma arbitrária e sobre esta solução são executadas operações com a intenção de melhorar sua qualidade, respeitando restrições do problema e finalizando o processo quando não é mais possível obter uma melhora [5].

Um exemplo de metaheurísticas são as baseadas em solução única (MBSU). Elas melhoram apenas uma única solução por iteração. As MBSU aplicam iterativamente procedimentos para geração e melhoria de uma solução para o problema, que substitui a solução atual, caso ela seja melhor. Conforme apresentado no Algoritmo 1, na fase de geração, um conjunto de soluções candidatas é gerado a partir da atual, geralmente obtidas por transformações locais. Na fase de substituição ocorre a seleção de uma solução do conjunto criado na fase anterior. Algo parecido como criar "passos" pela vizinhança enquanto "caminha" pelo espaço de busca do problema. Os passos são realizados por procedimentos iterativos que criam uma solução nova a partir da atual. A solução escolhida substituirá a solução atual, caso seja melhor. Este processo ocorre até que um critério de parada seja atingido.

Essas metaheuristicas mostraram sua eficiência em vários problemas de otimização de vários domínios, como por exemplo, eletrônica, aerodinâmica, dinâmica de fluídos, telecomunicações, aprendizado de máquina, modelagem de sistemas, processamento de imagens, planejamento de rotas, logística, entre outros [5].

Algoritmo 1 Modelo de um algoritmo Metaheurístico Baseado em Solução Única.
Adaptado de [5]

1: $S = gerarSolucaoInicial()$
2: T = 0
3: **repeat**
4: $C_T = gerarCandidatos(S)$
5: $S = selecionarSolucao(C_T)$
6: $T = T + 1$
7: **until** Critério de parada não for satisfeito

A metaheurística Greedy Randomized Adaptive Search Procedures (GRASP) usa uma heurística gulosa aleatória e iterativa para resolver problemas de otimização. Cada iteração do GRASP contém dois passos: construção e busca local. Conforme apresentado no Algoritmo 2, no passo de construção, uma solução possível é construída utilizando um algoritmo guloso aleatório. Uma heurística gulosa é aquela que sempre escolhe a melhor opção aparente, sem se preocupar se ela será também a melhor escolha para a solução global do problema. Entretanto, o GRASP utiliza um certo grau de aleatoriedade na construção da solução gulosa para poder diversificá-la. No passo seguinte, a heurística de busca local é aplicada na solução gerada. Esses dois passos são executados até que um critério de parada seja alcançado e a melhor solução encontrada seja definida como solução final [5].

Algoritmo 2 Modelo de um algoritmo GRASP.
Adaptado de [5]

1: **repeat**
2: $s = randomGreedy(seed)$
3: $s' = buscaLocal(s)$
4: **if** $s_{melhor} < s'$ **then**
5: $s_{melhor} = s'$
6: **end if**
7: **until** Critério de parada não for satisfeito

O GRASP possui um parâmetro chamado $\alpha \in [0, 1]$. Ele define o compromisso entre a diversificação e a intensificação. A diversificação induz a heurística a ampliar a exploração do espaço de busca, enquanto que a intensificação induz a heurística a direcionar a busca para a melhor solução local. Portanto, na diversificação o algoritmo será "mais aleatório", já na intensificação o algoritmo será "mais guloso". Assim, para $\alpha = 1$ a heurística gulosa se torna determinística, enquanto que para $\alpha = 0$ ela se torna aleatória.

Na fase de construção, a cada iteração, elementos que podem ser inseridos na possível solução serão ordenados (para isto, é necessário definir um valor numérico que representa a solução) em uma lista usando uma heurística local. Dessa lista, um subgrupo é escolhido, que representa a *restricted candidate list* (RCL). A seguir, um elemento aleatório é pego da lista RCL. Quando um elemento é incorporado na solução, a RCL é atualizada e os valores dos elementos na lista serão recalculados. Na fase de busca local, tradicionalmente é utilizado uma busca local simples, mas nada impede que outros métodos sejam utilizados, como, por exemplo, Busca Tabu, *Simulated Anneling*, etc. Nessa fase ocorre o melhoramento da solução gerada na fase anterior. O processo de melhoramento ocorre enquanto for possível realizar melhorias na solução.

C. Path Relinking

O *Path Relinking* explora os caminhos que conectam os vértices de soluções eficientes, chamadas de soluções de elite, para substituir partes da solução corrente por partes das soluções de elite. Portanto, a ideia é melhor explorar o espaço de busca por meio de um operador de intensificação que utilize soluções de elite já construídas. Dessa forma, parte-se de uma solução inicial e a partir das substituições de parte

dos caminhos tenta-se chegar a uma melhor solução final. Ou seja, o caminho entre duas soluções no espaço de busca geralmente vai criar soluções que compartilham atributos com as soluções de entrada. A sequência de soluções vizinhas será gerada a partir da solução inicial até alcançar a solução objetivo. A melhor solução gerada é retornada. A seleção do caminho considera os fatores importantes para a geração do caminho. A qualidade do processo de escolha do caminho pode interferir no nível de intensificação e diversificação das soluções geradas. Escolher as melhores ou as piores soluções terá um impacto diferente na busca [5].

Como o *Path Relinking* utiliza um ponto inicial e um final, várias estratégias podem ser utilizadas para explorar o caminho entre as soluções. Na estratégia *Forward*, a solução inicial sempre vai ser pior que a solução final. Na *Backward*, a inicial sempre vai ser melhor que a final. Na estratégia *Backward and Forward*, dois caminhos são construídos em paralelo, um caminho utiliza *Forward* e o outro *Backward*. Essa estratégia tem como característica ser necessário mais tempo de computação, que deve ser balanceado com a qualidade que pode ser obtida. Por último, a estratégia *Mixed* também constrói dois caminhos em paralelo, mas utiliza uma solução intermediária que está a mesma distância da solução inicial e final [5].

III. Trabalhos relacionados

O TOP foi apresentado pela primeira vez em [1], como um problema de gerenciamento de equipe em uma competição esportiva de trilha ao ar livre. A equipe tem que visitar várias localidades antes de finalizarem o percurso e receberem uma pontuação pelo trajeto de todos os membros. No trabalho foi proposta uma *heurística de duas etapas*: na primeira é construída uma elipse onde os vértices inicial e final são os focos da elipse e o tempo limite é o valor do maior eixo. Para construir o caminho, são considerados apenas os vértices dentro da elipse e são adicionados nas rotas seguindo uma heurística gulosa. Na segunda fase são realizadas trocas de vértices entre as rotas com a intenção de maximizar a pontuação da equipe. Os autores aplicaram a heurística em 353 problemas que variam de 21 até 102 vértices e de 2 a 4 rotas, conseguindo resultado computacionalmente eficiente.

Em [6], desenvolveu-se um algoritmo baseado em *Particle Swarm Optimization* (PSO). A ideia do algoritmo PSO é simular o comportamento coletivo de animais selvagens na natureza. Assim, o comportamento do PSO consiste em uma população de partículas, que são possíveis soluções do problema, iterada atualizando a posição atual de todas as partículas levando em consideração a melhor solução local e global. No trabalho [6], o algoritmo foi testado em 387 instâncias, com 2 a 4 rotas e se mostrou competitivo tanto em tempo, como em qualidade da resposta.

No trabalho [7], introduziu-se uma versão do problema TOP com multi-restrições e janela de tempo. Um conjunto de localidades é dada, onde cada uma possui uma tarefa com tempo de execução, uma ou mais janelas de tempo e uma pontuação. O objetivo é maximizar a soma da pontuação, em um número fixo de rotas. As rotas são limitadas por comprimento e restritas com janelas de tempo e adicionais restrições. Um algoritmo hibrido, integrando *busca local iterada* e um procedimento de *busca gulosa adaptativa randomizada* foi proposto. O experimento envolveu 148 instâncias com 48 a 288 localizações, 1 a 4 rotas, 11 tipos de restrições e múltiplas janelas de tempo por localização. Resultados dos experimentos alcançaram o estado da arte conhecido em apenas 5.19% das execuções, utilizando como máximo de execução 1.5 segundos. Mas em instâncias de testes, obtiveram 32% de resultados iguais aos do estado da arte.

Voltando para o problema original, em [8], foi proposta uma abordagem baseada em *Branch-and-Price*. O algoritmo foi testado em 387 instâncias agrupadas em sete conjuntos. Os conjuntos apenas diferem em número de veículos e tempo máximo. O algoritmo resolveu 80% das instâncias, incluindo 5% ainda não resolvidas. Já no trabalho [9], foi proposto um algoritmo baseado *multi-start simulated anneling* juntamente com estratégia *multi-start hill climbing*. Nele, testaram-se 157 instâncias, sendo possível alcançar o estado da arte em 135 delas sendo que em 5 instâncias os resultados obtidos transformaram-se em novos estados da arte. Em [10] foi investigada uma estratégia de formulação linear com número polinomial de variáveis. *Cutting planes* é o componente principal do algoritmo. Para testes utilizaram-se 387 instâncias, que variam de 19 a 100 localizações, com 2 a 4 rotas. Resultados indicaram a resolução de 300 das 387 instâncias.

Dos seis trabalhos apresentados, um utilizou uma heurística própria, dois utilizaram métodos exatos de resolução, dois utilizaram metaheurísticas baseadas em solução única e um utilizou algoritmo metaheurístico baseado em população. Como pode ser observado pela revisão apresentada, a utilização de metaheurísticas de melhoramento de solução única se apresentam como abordagem adequada para a aplicação nesse tipo de problema.

Importante notar que dentre as abordagens apresentadas, apesar da aplicação de metaheurísticas baseadas em solução única, nenhum trabalho apresentou estudo envolvendo técnica de busca adaptativa aleatória com intensificação de resultados propiciada por reutilização de parte de soluções de elite do problema. Este trabalho visa apresentar proposta nesse sentido, cobrindo tal lacuna nos estudos relativos à aplicação do TOP em aplicações logísticas.

IV. Proposta de solução

Esta seção apresenta detalhes da lógica de implementação da metaheurística GRASP para a solução de roteamento utilizando a abordagem TOP.

A. Fase de geração

No TOP, cada instância do problema tem um número de rotas que deve ser gerada. O primeiro vértice de todas as rotas é sempre o mesmo. Conforme apresentado no Algoritmo 3, na primeira rota, cada vértice que pode ser visitado vai receber um valor que depende do valor da distância até o último vértice da rota e a recompensa. Esse valor é calculado pela Eq. 9, onde

sd_i é o valor com relação a distância entre o vértice candidato e o último vértice na rota, e sr_i é o valor com relação a recompensa que será coletada se a visita ocorrer.

Os valores de recompensa e distância são calculados de acordo com as Equações 10 e 11, respectivamente. Essas equações foram utilizadas para normalizar os dados de diferentes intervalos. Esse método foi apresentado em [11]. Logo, um vértice que possua uma recompensa grande e que esteja próximo é interessante para a rota. Grande recompensa, mas distante, é menos atraentes. Pouca recompensa e muito distante é pouco atraente. Portanto, a ideia é privilegiar a seleção dos melhores elementos, porém, não deve impedir a seleção de piores. Este trabalho vai adotar a **escolha probabilística** para realizar a inserção de vértices em rotas.

Algoritmo 3 Algoritmo da fase de geração

repeat
  for Cada rota r_j **do**
    for Para cada vértice v_i **do**
      $calculaValorVertice(v_i)$ // $s_i = 1.1 + sr_i - sd_i$
    end for
    for Para cada vértice v_i **do**
      $calcProbabilidadeVertice(v_i)$ // $p(i) = \frac{s_i}{\sum_p^{j=1} s_j}$
    end for
    $v = sortearVertice(alpha)$ // Escolha probabilística
    $r_j = r_j \bigcup v$
  end for
until Pelo menos uma rota recebeu a adição de um vértice

Após o cálculo alguns vértices terão valores altos e outros baixos. O algoritmo então fará a seleção com a escolha probabilística. Vértices com valores altos terão probabilidades maiores de serem selecionados, mas não existe a garantia que eles serão selecionados. Assim, como existe a chance de um vértices ruim ser escolhido, a probabilidade de seleção de um vértice é dado pela Equação 12. Após a seleção, os valores para os vértices considerando a próxima rota são recalculados. É importante notar que a inserção do vértice na rota está condicionada a restrição de tempo máximo que a rota possui. O algoritmo finaliza quando não for possível mais inserir nenhum vértice nas rotas, devido a restrição de tempo.

$$s_i = 1.1 + sr_i - sd_i \tag{9}$$

$$sr_i = (r_i - r_{min})/(r_{max} - r_{min}) \tag{10}$$

$$sd_i = (d_i - d_{min})/(d_{max} - d_{min}) \tag{11}$$

$$p(i) = \frac{s_i}{\sum_p^{j=1} s_j} \tag{12}$$

B. Fase de busca local

A Busca Local Simples foi o método implementado como operador de busca local no GRASP neste trabalho. Ele foi selecionado por ser de fácil implementação e pelo potencial de identificação de ótimos locais. Ainda, é importante indicar quais operadores de geração de vizinhança foram utilizados nesta fase. O primeiro operador realiza a troca de vértices dentro de uma mesma rota. Ele tem a intenção de diminuir o tempo total da rota. O segundo realiza a inserção de novos vértices na rota. Esse tenta aumentar a recompensa da solução, ou seja, incluir a maior recompensa possível dentro de cada rota. O terceiro realiza a sobrescrita de vértices na rota e também tenta aumentar a recompensa da rota. O Algoritmo 4 apresenta a aplicação desses operadores.

Algoritmo 4 Algoritmo da fase de busca local

input: s // solução válida
$s_{melhor} = s$
repeat
  $s_i = trocarVerticeNaMesmaRota(s_i)$
  $s_i = adicionarNovoVertice(s_i)$
  $s_i = sobrescreverVertice(s_i)$
  if $s_{melhor} < s$ **then**
    $s_{melhor} = s$
  end if
until Critério de parada não for satisfeito

O operador de troca na mesma rota realiza trocas de dois vértices na mesma rota até não haver mais trocas possíveis. A troca que mais diminuir a distância da rota vai ser selecionada. O processo ocorre em todas as rotas da solução. A melhor troca em todas as rotas é salva. Um exemplo está na Fig. 1. A rota original é (0,1,2,3,4,0). Na primeira iteração, os valores 1 e 2 são trocados entre si, gerando a solução (0,**2**,**1**,3,4,0). Na próxima iteração ocorre a troca dos valores 1 e 3, gerando (0,**3**,2,**1**,4,0), e assim por diante, gerando as soluções (0,**4**,2,3,**1**,0), (0,1,**3**,**2**,4,0), (0,1,**4**,3,**2**,0) e (0,1,2,**4**,**3**,0). É importante mencionar que o valor zero na rota representa o vértice inicial e final da rota.

Figura 1. Operador de troca

O operador de adição tenta adicionar um novo vértice na solução. Ele inicia pela primeira posição da primeira rota,

seguindo para a segunda posição, e assim por diante, até o final da rota. Ao término de uma rota, o processo se inicia na próxima rota. Quando a adição gerar uma solução com recompensa maior, esta será retornada pelo operador. Considerando a rota de exemplo (0,1,2,3,4,0) e o vértice de valor 5, que estão na Fig. 2, é possível criar a seguinte sequência de rotas: (0,**5**,1,2,3,4,0), (0,1,**5**,2,3,4,0), (0,1,2,**5**,3,4,0), (0,1,2,3,**5**,4,0) e (0,1,2,3,4,**5**,0).

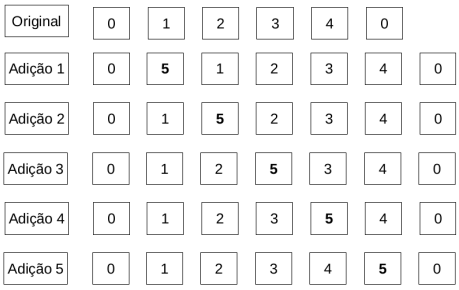

Figura 2. Operador de Adição

O operador de sobrescrita funciona de forma análoga ao de adição. Entretanto, ao invés de adicionar um vértice, ele sobrescreve um vértice na rota. Ele inicia pela primeira posição da primeira rota, seguindo para a segunda posição, e assim por diante, até a finalizar a rota. Ao término de uma rota, a próxima é selecionada. Este operador também retorna a primeira solução que aumenta a recompensa. Considerando a rota de exemplo (0,1,2,3,4,0) e o vértice de valor 5 (Fig. 3), é possível criar a seguinte sequência de rotas: (0,**5**,2,3,4,0), (0,1,**5**,3,4,0), (0,1,2,**5**,4,0) e (0,1,2,3,**5**,0).

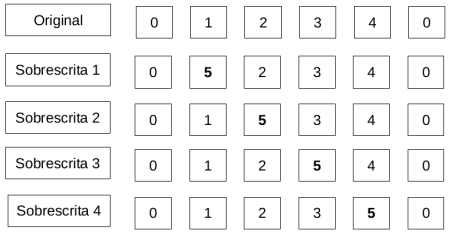

Figura 3. Operador de sobrescrita

C. Path Relinking

O funcionamento básico do ***Path Relinking*** consiste em atribuir características de uma solução, chamada inicial, à outra solução, chamada final. A solução resultante pode ser a inicial, a final ou uma que tenha características de ambas. O último caso significa que foi encontrado uma terceira solução melhor que a inicial e a final. Neste trabalho, o processo de passar as características de uma solução para outra consiste em sobrescrever uma solução com características da outra. O Algoritmo 5 demostra o funcionamento do *Path Relinking*. A sobrescrita inicia pela primeira posição da primeira rota, seguida da segunda posição da primeira rota, e assim por diante. Quando uma rota chega ao final, o processo inicia na rota seguinte.

Algoritmo 5 Algoritmo de melhoramento com *Path Relinking*

1: **input**: $s_{inicial}$, s_{final}
2: $s_{melhor} = s_{inicial}$
3: $pos = 0$
4: **repeat**
5: $s = sobrescreverSolucao(s_{inicial}, s_{final}, pos)$
6: $pos = pos + 1$
7: **if** $s_{melhor} < s$ **then**
8: $s_{melhor} = s$
9: **end if**
10: **until** $pos <= tamanho(s_{inicial})$

Para um melhor entendimento, a seguir é apresentado um exemplo na Fig. 4. A solução inicial é representa pelos vértices (0,1,2,3,4,0) e a solução final pelos vértices (0,5,6,7,8,0). Na primeira iteração, a primeira posição da solução final sobrescreveu a primeira posição da solução inicial, gerando a solução inicial (0,**5**,2,3,4,0). Na segunda iteração será (0,**5**,**6**,3,4,0), onde a segunda posição é sobrescrita. Na próxima solução a terceira posição é sobrescrita (0,**5**,**6**,**7**,4,0) e na última solução a quarta posição é sobrescrita (0,**5**,**6**,**7**,**8**,0). As soluções inicial e final não são alteradas durante o processo. Todas as sobrescritas são realizadas em uma terceira solução.

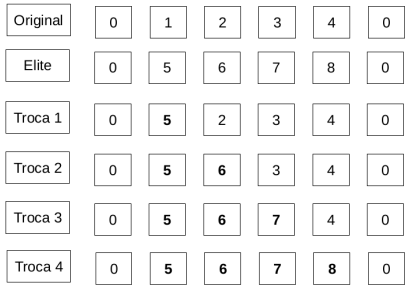

Figura 4. Operador de Path Relinking

V. Experimentos e Resultados

Nesta seção são apresentados os resultados alcançados, começando pela apresentação da metodologia, seguida pela execução dos experimento e finalizando com a análise dos resultados.

A. Metodologia de Avaliação

O algoritmo foi implementado na linguagem C++, compilador g++ versão 7.5.0. O sistema operacional utilizado foi o Ubuntu 18.04.5 LTS com Intel Core i5-8265U 1.60GHz de 8 núcleos e 8 Gb de RAM. O algoritmo implementado foi carregado na ferramenta `irace` [12] para descobrir qual a melhor combinação de parâmetros. Em posse dos melhores parâmetros foi possível realizar os experimentos e coletar os resultados. O algoritmo implementado foi executado sobre o conjunto de

instâncias apresentado em [1]. Cada instância foi executada 30 vezes, com coleta de tempo e qualidade da solução apresentada pelo algoritmo proposto em cada execução. O objetivo foi verificar o potencial de otimização alcançado pelo algoritmo proposto nesse trabalho em comparação com outros propostos na literatura.

B. Calibração de parâmetros

Com o algoritmo implementado foi executado em uma ferramenta chamada `irace`. Esta ferramenta automatiza a configuração de parâmetros e define o melhor conjunto possível baseada em execuções do algoritmo. O `irace` recebe todos os parâmetros e uma especificação do tipo e intervalo de valores possíveis. Ao terminar o processo, fornece combinações que obtém os melhores resultados.

Os parâmetros utilizados são:

- **alpha**: parâmetro do GRASP que indica se será priorizada a intensificação ou a diversificação. O intervalo de valores é $[0, 1]$.
- **iterations**: indica quantas iterações o algoritmo deve executar antes de parar e apresentar a melhor~~a~~ solução encontrada. O intervalo de valores é $[100, 5000]$.
- **path**: parâmetro do *Path Relinking* que indica o sentido da sobrescrita das soluções. Pode ser no sentido inicial para a final ou da final para a inicial.

A ferramenta ainda precisa de instâncias e um número de experimentos. As instâncias foram selecionadas na proporção de 10% de todas as instâncias utilizadas no experimento final. O número de testes foi 3000. O resultado final apresentado foi alpha de $0,86$, iterations de 3540 e o parâmetro *path* de inicial para a final.

C. Resultados

A Tabela I apresenta os resultados obtidos no experimento. A primeira coluna indica o conjunto de instâncias. A segunda coluna mostra o número de instâncias. A terceira indica o número de rotas no conjunto. A quarta coluna mostra o número de vértices nas instâncias. A quinta coluna indica em quantas instâncias o algoritmo proposto chegou pelo menos uma vez no estado da arte (EDA). A sexta coluna indica em quantas instâncias a média das 30 execuções ficou igual ao estado da arte. Os valores de estado de arte foram retirados de [13].

Das 277 instâncias usadas, o algoritmo proposto conseguiu chegar no estado da arte em todas as trintas execuções em 105 instâncias (37,9%). O algoritmo também conseguiu chegar no estado da arte em pelo menos uma vez em outras 24 instâncias (8,6%). Somando os dois casos, totaliza 46,5% das instâncias. Porém, nas maiores instâncias (conjuntos P4.2, P4.3 e P4.4, com 100 vértices por instâncias), por exemplo, em apenas 17,8% das instâncias o algoritmo alcançou ao menos uma vez o estado da arte.

Tabela I
RESULTADOS DO EXPERIMENTO

Conjunto	Instâncias	Rotas	Vértices	EDA	Média no EDA
P1.2	18	2	32	2	8
P1.3	18	3	32	6	8
P1.4	18	4	32	4	12
P2.2	11	2	21	0	11
P2.3	11	3	21	0	11
P2.4	11	4	21	0	11
P3.2	20	2	33	0	5
P3.3	20	3	33	1	6
P3.4	20	4	33	0	9
P4.2	20	2	100	0	1
P4.3	19	3	100	2	2
P4.4	17	4	100	1	4
P5.2	25	2	66	2	4
P5.3	25	3	66	3	6
P5.4	24	4	66	3	7
Total	277			24	105

VI. CONCLUSÃO

Este artigo propôs um algoritmo baseado em GRASP e *Path Relinking* para o problema *Team Orienteering Problem*. O problema consiste em montar um conjunto de rotas que propiciam o atendimento/execução de tarefas em locais fisicamente diferentes. Apenas as tarefas que respeitem a restrição de tempo são possíveis de serem atendidas. A implementação do algoritmo utilizou-se da linguagem C++ e experimentos foram realizados utilizando as instâncias de teste disponíveis em [1]. O experimento executado com o algoritmo proposto atingiu o estado da arte em 46,5% das instâncias. Entretanto, por se tratar de um trabalho em desenvolvimento, espera-se que durante sua evolução esses resultados apresentem melhoras.

Como trabalhos futuros, sugere-se implementar outros operadores de busca local para melhorar o desempenho do algoritmo. Outra abordagem possível é verificar o impacto do *Path Relinking* na qualidade da solução e averiguar que outras heurísticas de geração de soluções poderia impactar positivamente no resultado.

REFERÊNCIAS

[1] I.-M. Chao, B. L. Golden, and E. A. Wasil, "The team orienteering problem," *European journal of operational research*, vol. 88, no. 3, pp. 464–474, 1996.

[2] S. E. Butt and T. M. Cavalier, "A heuristic for the multiple tour maximum collection problem," *Computers & Operations Research*, vol. 21, no. 1, pp. 101–111, 1994.

[3] H. Tang and E. Miller-Hooks, "A tabu search heuristic for the team orienteering problem," *Computers & Operations Research*, vol. 32, no. 6, pp. 1379–1407, 2005.

[4] L. Ke, C. Archetti, and Z. Feng, "Ants can solve the team orienteering problem," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 648–665, 2008.

[5] E.-G. Talbi, *Metaheuristics: from design to implementation*. United States: John Wiley & Sons, 2009, vol. 74.

[6] D.-C. Dang, R. N. Guibadj, and A. Moukrim, "An effective pso-inspired algorithm for the team orienteering problem," *European Journal of Operational Research*, vol. 229, no. 2, pp. 332–344, 2013.

[7] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden, "The multiconstraint team orienteering problem with multiple time windows," *Transportation Science*, vol. 47, no. 1, pp. 53–63, 2013.

[8] M. Keshtkaran, K. Ziarati, A. Bettinelli, and D. Vigo, "Enhanced exact solution methods for the team orienteering problem," *International Journal of Production Research*, vol. 54, no. 2, pp. 591–601, 2016.

[9] S.-W. Lin, "Solving the team orienteering problem using effective multi-start simulated annealing," *Applied Soft Computing*, vol. 13, no. 2, pp. 1064–1073, 2013.

[10] R. El-Hajj, D.-C. Dang, and A. Moukrim, "Solving the team orienteering problem with cutting planes," *Computers & Operations Research*, vol. 74, pp. 21–30, 2016.
[11] J. Han, M. Kamber, and J. Pei, "Data mining concepts and techniques third edition," *The Morgan Kaufmann Series in Data Management Systems*, vol. 5, no. 4, pp. 83–124, 2011.
[12] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package, iterated race for automatic algorithm configuration," Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles ..., Tech. Rep., 2011.
[13] H. Bouly, D.-C. Dang, and A. Moukrim, "A memetic algorithm for the team orienteering problem," *4or*, vol. 8, no. 1, pp. 49–70, 2010.