Projeto Final de Programação
INF2102

Um algoritmo Branch-and-Price para roteamento de ambulâncias

André Mazal Krauss

Orientador: Prof Thibaut Vidal

06 de Dezembro de 2022

Introdução

O presente documento descreve e documenta o programa StaticAmbulanceVRP disponível em [1] e desenvolvido por André Mazal Krauss no processo de obtenção de sua tese de Mestrado. O programa consiste em um algoritmo Branch-and-Price para a resolução de um problema estático de roteamento de ambulâncias. Serão definidas neste documento as especificações do programa em questão, bem como o seu modo de uso. Além disso será documentada a implementação do programa, feita na linguagem C++ usando o framework SCIP [2] para programação linear-inteira. Porém, detalhar o problema e a metodologia usada está fora do escopo deste documento. Estes temas são abordados na tese de Mestrado que acompanha este projeto.

Especificação do Programa

O programa deve atender aos seguintes requisitos:

- Realizar leitura de arquivos com os dados de entrada necessários e escrita de arquivos com as saídas do modelo.
- Implementar o modelo *Branch and Price* descrito na tese de mestrado que acompanha este documento, assim como a comunicação com o solver SCIP / SoPlex e o algoritmo auxiliar que identifica novas variáveis a serem adicionadas ao solver
- Disponibilizar ao usuário acesso às funcionalidades implementadas através de interface de linha de comando.

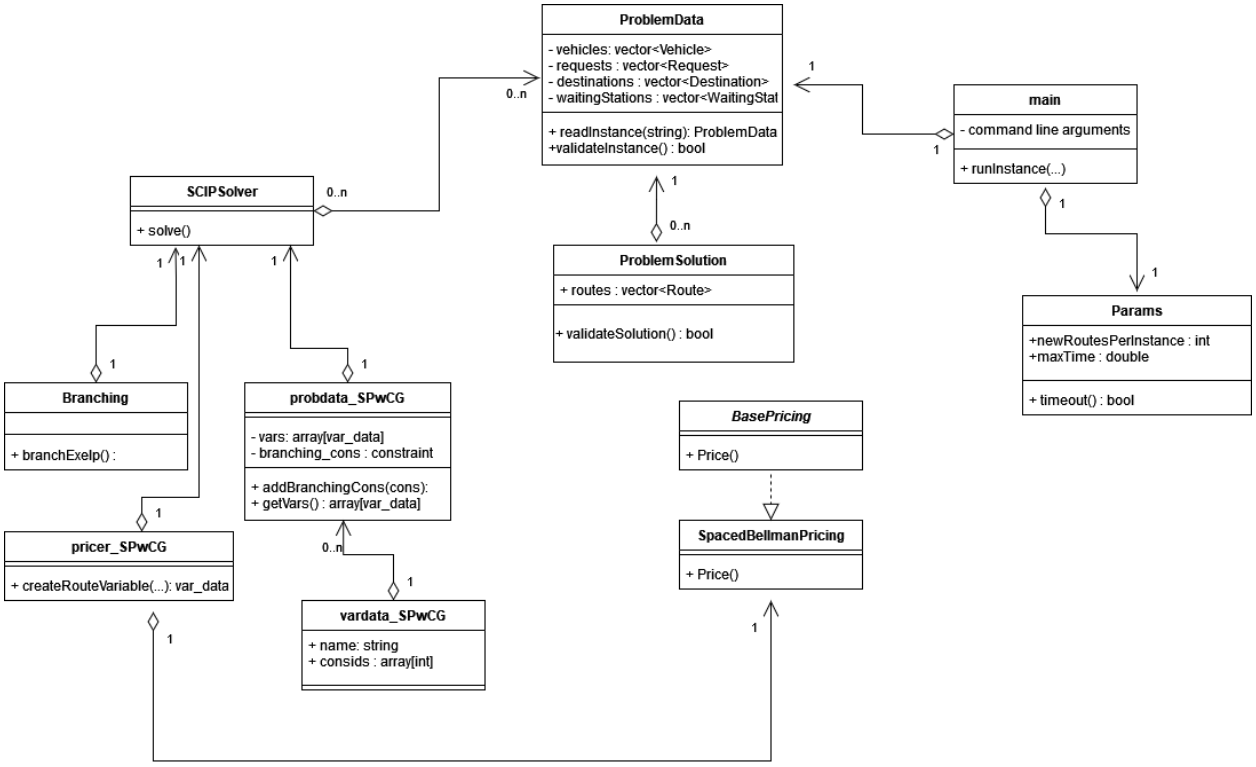
Implementação e Arquitetura

A implementação do programa é feita usando C++17 e o framework SCIP de otimização linear inteira. O solver SCIP é um software de código aberto e não comercial, desenvolvido no Zuse Institut Berlin, em Berlim, Alemanha. Para mais informações sobre o SCIP, ver [1].

A arquitetura de software segue o diagrama UML abaixo, com as seguintes classes:

- Main - implementa a interface de linha de comando e inicializa execuções do modelo
- ProblemData - classe responsável por armazenar os dados relativos a uma instância do problema, assim como realizar a leitura dos arquivos de onde são carregados os dados.
- ProblemSolution - classe responsável por armazenar uma solução a uma instância.
- Params - responsável por armazenar meta-parâmetros relativos à opções de execução do modelo.

- OSRMHelper - Implementa a comunicação com a biblioteca osrm-backend, usada para calcular a distância entre duas coordenadas geográficas baseada na malha rodoviária da região de estudo. A malha deve ser fornecida como uma base de dados no formato Open Street Map, já preprocessada no modo Multi Level Dijkstra.
- RouteExpander - Implementa as regras de expansão de rotas. A expansão deve respeitar as regras de expansão definidas no problema, por exemplo, não-antecipatividade e compatibilidade entre veículos e chamadas, opções sobre uso de estações de espera etc. São calculados o tempo e pênaltis resultantes da expansão.
- SCIPSolver - Implementa a inicialização, configuração e comunicação com o solver SCIP.
- BasePricing - Classe abstrata definindo a interface para implmentação de algoritmos *pricing*, ou seja, algoritmos para obtenção de novas variáveis com custo reduzido negativo.
- SpacedBellmanPricing - classe responsável pela implementação do algoritmo *pricing* usado no trabalho em questão
- Branching - Implementa as as regras de *branching* para o algoritmo *Branch and Price*
- probdata_SPwCG - Define dados e implementa funções auxiliares usadas na comunicação com o solver
- pricer_SPwCG - Define dados e implementa funções auxiliares usados no contexto da comunicação com o *solver* no que diz respeito ao *pricing*.
- vardata_SPwCG - Define dados e implementa funções auxiliares usados no contexto da criação de novas variáveis



Certos relacionamentos e atributos foram omitidos por questões de clareza de leitura. Em particular, diversos elementos tem uma relação com a classe Params, que funciona essencialmente como uma referência global. As classes Branching, probdata_SPwCG, pricer_SPwCG e vardata_SPwCG estão relacionadas à certas obrigações que o a interface do SCIP impõe. Para maiores detalhes, recomenda-se estudar os projetos exemplos em [1], em particular o Binpacking. Além disso, algumas estruturas de dados relativas à definição do problem foram omitidas. São elas *Vehicle*, *Request*, *Destination*, e *WaitingStation*. Elas tem uma correspondência direta com as entidades descritas na tese de mestrado associada a esse projeto, sendo equivalentes a veículos, chamadas, destinos e estações de espera respectivamente. Recomenda-se ler a seção *Problem Definition* da tese para maiores detalhes.

Uma descrição mais detalhada do código-fonte está disponível no repositório do projeto.

Usuário Final e Como Usar

O usuário-alvo do programa em questão é um pequeno grupo de pesquisa liderado pelos orientadores da tese de mestrado anexa à esse documento. Em particular, a principal meta em seu desenvolvimento é dar base experimental ao algoritmo apresentado na tese. Dito isso, foi feita uma interface por linha de comando, ferramenta familiar ao grupo em questão, e que permitiu a automatização dos testes realizados para a pesquisa de tese.

Descreve-se agora a interface de uso do programa por linha de comando. O projeto foi desenvolvido usando CMake e pode ser usado em Windows e Linux, quando compilado usando CMake e qualquer compilador compatível com C++17. Ainda assim, recomenda-se utilizar Ubuntu 20.14 e o compilador GCC.

Os argumentos de entrada definem as entradas ao solver. Ou seja, aqueles que controlam qual instância deve ser solucionada e com que variações de roteamento, cálculo de distâncias etc. São eles:

- type (string) - tipo de arquivo de entrada a ser lido. Opções: “PDPTW”, “SDVRPTW”, ‘V’, “V10”. Default “V”
- path (string) - , caminho ao diretório / arquivos da instância, para os tipos “PDPTW”, “SDVRPTW”, ‘V’, “V10”
- Requests_path (string) - caminho para o arquivos de chamadas. Usado com instâncias “V”.
- V_index (int) - índice da instância no arquivo de chamadas. Usado somente com instâncias do tipo “V”
- osmPath(string) - Caminho opcional para o diretório de dados Open Street Map. Valor default vazio.
- instance_index (int) - argumento auxiliar para selecionar uma instâncias diretamente por um índice relativo às instâncias usadas na tese, dado que o argumento path recebeu um diretório com as instâncias da tese.

- time_horizon_usage (string) - definições sobre o uso de um horizonte de tempo. Na opção “default” é usado um horizonte padrão dependente da instância lida. “'Infinite” usa um horizonte infinito. “'Value” usa o valor passado no argumento “time_horizon”. “compute” computa dinamicamente um horizonte mínimo para garantir atendimento a todas as chamadas. Valor default é “default”.
- time_horizon (double) - horizonte de tempo a ser usado quando se usa a opção “time_horizon” para o argumento time_horizon_usage.
- waitingStationPolicy (int) - define a regra usada na expansão de rotas, em se tratando do uso de estações de espera. “0” indica parada obrigatória entre chamadas, em uma estação fixa. “1” indica uma parada opcional em uma estação fixa. “2” indica uma parada opcional na estação mais próxima do local de término do último atendimento. “3” indica que o solver deve buscar a melhor parada dentre todas as possibilidades. Valor default 0.
- allow_rerouting (int) - Determina se reroteamentos de estações de espera para chamadas serão permitidos. “0” para não, “1” para sim. Valor default “0”.
- use_target_objective (int) - determina se o solver deve usar o objetivo alternativo de minimizar o número de chamadas fora do tempo objetive. “0” para não, “1” para sim. Valor default “0”.
- set_nb_vehicles (int) - sobrescreve o número usual de veículos na instância para o numero passado como argumento.
- relaxed (int) - Determina se o solver deve rodar uma versão relaxada linearmente do problem. “0” para não, “1” para sim. Valor default “0”
- Heuristic_run (int) - Determina se deve ser usada uma heurística em vez do solver exato. “0” para não. “1” para heurística *Closest Available*. “2” para heurística *Earliest Arrival*. Valor default “0”

Os argumentos de saída definem como deve ser feito o output da solução obtida pelo solver. São eles:

- output_dir (string) - diretório de saída para os arquivos de logging e output. Valor default “*./*”.
- Output_suffix (string) - determina opcionalmente um sufixo a ser adicionado aos arquivos de saída, para facilitar sua posterior identificação.
- outputDuals (int) - Determina se o solver deve exportar arquivos contendo os valores das variáveis duais ao fim da execução. “0” para não, “1” para sim, valor default “0”.
- descriptiveString (string) - uma string opcional para ser adicionada ao nome da instância nos arquivos de output.

Os argumentos de meta-parametros definem parâmetros que controlam a execução do problema. São eles:

- max_time (double) - tempo máximo em segundo permitidos para a execução de todo o algoritmo. Valor default 1e20+.
- max_pricing_time(double) - tempo máximo permitido para a execução de uma rodada do algoritmo de pricing. Valor default 1e20+.

- max_memory (double) - valor máximo de memória, em MB, permitida para alocação do SCIP. Valor default 10000.0.
- max_pricing_memory - valor máximo de memória, em MB, permitida para alocação do algoritmo pricing. Valor default 10000.0.
- New_routes_per_pricing (int) - determina quantas novas variáveis, no máximo, são adicionadas a cada rodada do algoritmo pricing. Valor default 10.
- N_random_initial_routes (int) - determina quantas rotas geradas aleatoriamente devem ser inseridas no nó raiz, no início da execução. Valor default 0.
- Route_gen_seed (int) - determina a seed para geração de valores aleatórios. Valor default 0.

Testes

Os testes foram realizados com 120 instâncias sintéticas fabricadas usando dados do SAMU da cidade do Rio de Janeiro, Brasil. Para mais detalhes sobre a geração das instâncias, recomenda-se consultar a tese de Mestrado que acompanha este documento. De formar geral, podemos dividir os testes em três etapas.

Primeiramente, foram realizados testes de análise estática e testes automatizados de execução de código. Esses testes analisam somente a coerência e boa execução do programa, não identificando possíveis erros sensíveis aos dados do problema. Foi utilizada a ferramenta *valgrind.* O valgrind é capaz de detectar erros do tipo vazamento de memória, execução não-determinística, *undefined behaviour* etc. O valgrind foi utilizado ao longo do desenvolvimento do programa para se obter uma detecção rápida de erros corriqueiros introduzidos durante o desenvolvimento do código. Ao término do projeto, o valgrind reporta 0 erros e 0 warnings, quando rodado com todas as instâncias.

Em uma segunda etapa, foram implementados testes de sanidade em relação aos dados de entrada e saída do solver. Os testes verificam, por exemplo, se o número de rotas obtidas condiz com o número de veículos disponíveis, se as chamadas são atendidas de acordo com o seu tempo de chegada e as regras de roteamento, etc. Os testes foram implementados diretamente nos módulos *ProblemData* e *ProblemSolution*, e foram implementados de maneira que, ao detectar-se incoerência nos dados durante ou ao término de uma execução, o programa é abortado com uma mensagem de erro.

Por último, realizamos testes em série com todas as instâncias geradas, variando-se o horizonte de tempo, número de chamadas, número de veículos, região geográfica usada etc. Os resultados obtidos apresentam grande variabilidade no que diz respeito à tempo de execução, convergência, valor objetivo obtido, número de nós na árvore de resolução, número de variáveis, etc. Apresentamos alguns resultados aqui em forma de imagem, mas a descrição completa está reservada à tese.

Table 5.1: Results for default MIP by time horizon and region size

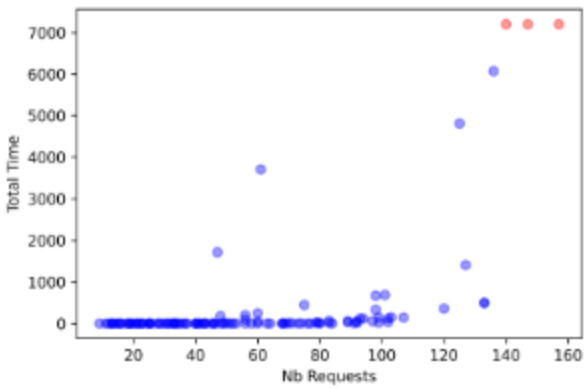
	Nb Reqs	Obj. Value	Penalty	Not Serviced	RT(m)	Weighted RT
t2_S	13.90	500.92	336.05	0.90	4.13	11.86
t4_S	25.00	1386.86	936.06	1.10	5.87	18.03
t6_S	38.70	1195.76	432.02	0.30	6.36	19.73
t8_S	52.60	1397.17	480.02	0.40	5.71	17.44
t2_M	19.30	283.03	108.02	0.60	3.34	9.07
t4_M	46.20	868.49	408.03	0.70	3.86	9.97
t6_M	73.70	1028.76	180.01	0.20	4.29	11.52
t8_M	95.00	2293.83	1296.04	0.90	4.04	10.50
t2_L	33.20	886.15	396.06	1.30	5.26	14.76
t4_L	66.50	1443.60	552.04	1.00	5.36	13.41
t6_L	89.10	2090.70	900.04	0.80	5.32	13.36
t8_L*	132.50	4210.33	1968.07	1.50	6.20	16.92

Table 5.2: Results for default MIP by time horizon and region size (continued)

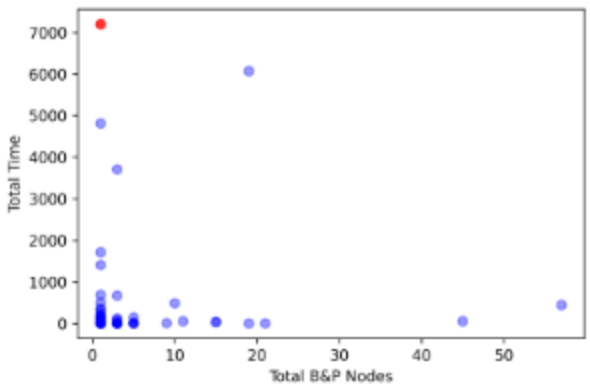
	Nb Routes	Runtime(s)	Nb Vars	Nb Nodes	Reqs/Routes
t2_S	9.60	0.02	85.60	1.00	1.36
t4_S	10.70	0.06	364.00	1.20	2.23
t6_S	12.00	189.87	1179.90	1.00	3.20
t8_S	12.00	432.88	2443.40	1.40	4.35
t2_M	15.10	0.02	110.70	1.00	1.23
t4_M	21.60	0.37	721.60	4.00	2.11
t6_M	23.60	54.64	2316.40	8.00	3.11
t8_M	24.30	242.57	3953.40	2.00	3.88
t2_L	23.00	0.04	235.30	1.60	1.38
t4_L	30.20	1.33	1048.30	3.40	2.16
t6_L	30.30	23.67	2487.70	9.60	2.91
t8_L*	33.40	3537.78	6607.40	3.70	3.92

Table 5.3: Aggregated results for 113 instances solved to optimality with variations

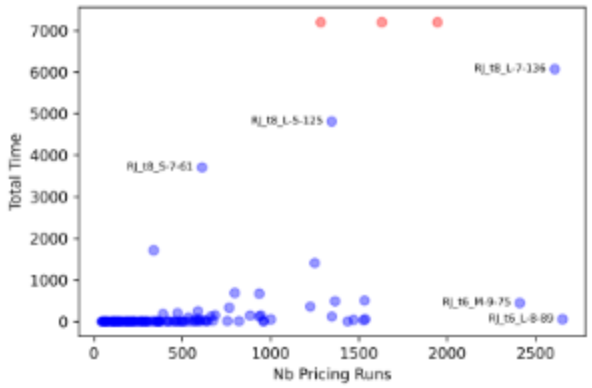
	Obj. Value	Penalty	RT(m)	Weighted RT
Online Closest Avail	2058.84	1047.14	6.14	18.95
Offline Mip	1314.45	609.59	4.90	13.20
Mip w fixed station	1495.61	716.86	5.20	14.59
Mip w Opt Reass	844.45	456.66	2.65	7.26
Mip w Redirection	1266.93	575.61	4.81	12.95
Mip w Opt Reass and Red	745.67	415.25	2.24	6.19



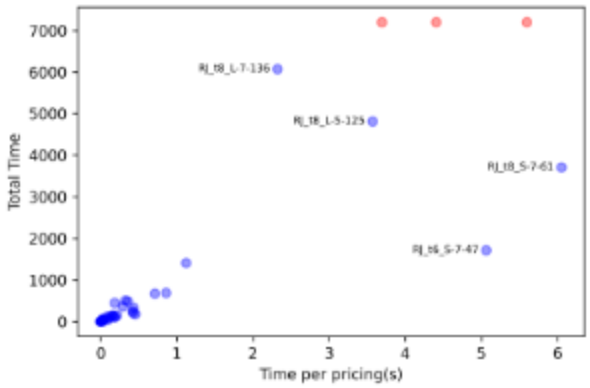
(a) Nb Requests X Time



(b) Nb Nodes X Time



(c) Nb Pricing Runs X Time



(d) Time per Pricing X Time

Figure 5.1: Comparison of different measures against computation time for default MIP runs

Referências

[1] - Repositório com código fonte e documentação do programa - https://github.com/amk1710/StaticAmbulanceVRP . Acessado em 06/12/2022
[2] - Framework SCIP para otimização linear-inteira https://scipopt.org/. Acessado em 06/12/2022