Procura e Planeamento

Campus da Alameda

Projeto (2018/2019)

Número do grupo (obtido a partir do sistema Fénix): 1

Nome: João Francisco Lopes Pirralha Número: 78375

Nome: Rafael Meneses Lucas Ribeiro Número: 84758

Classificação:

Soma das horas gastas exclusivamente para fazer este trabalho: 70

Limite para entrega: dia 23h59 do dia 7 de Dezembro de 2018.

Índice

Descrição do trabalho desenvolvido

Modelação do problema

Para modelar o problema usou-se um conjunto de tarefas alocadas em turnos (listas de tarefas) e outro conjunto de tarefas por alocar. De modo a tornar as operações sobre as tarefas mais eficientes, estas são apenas números que identificam a tarefa completa, no problema, que inclui toda a informação do *input*.

Um estado é composto, fundamentalmente, pelos dois conjuntos referidos. As restantes propriedades são detalhes de implementação e são explicitadas na sub-secção seguinte.

A geração dos sucessores de um estado corresponde a associar cada tarefa com cada um dos turnos existentes (se possível) e a criar um novo turno que apenas a contém. Como tal, verificar se um estado é objetivo resume-se a verificar se o conjunto de tarefas não alocadas é vazio. Dois estados são considerados iguais se o seu conjunto de tarefas alocadas for igual.

Estruturas de dados

Estados

Para modelar os estados do problema usou-se a estrutura representada pelo exemplo da Figura 1.

```
#S(AFECTACAO-STATE
   :PROBLEM #2A((2 1 1 25)
                (1 2 34 60)
                (5 1 408 447)
                (1 1 448 551)
                (1 1 474 565))
   :ALLOCATED ((4 2) (3) (0))
   :NON-ALLOCATED (1)
   :COST 0
   :TOTAL-COST 1080
   :HEURISTIC 0
   :HEURISTIC-FUNCTION #<FUNCTION GREEDY-ALLOCATION-HEURISTIC>
   :HASH 3165602969652788954)
```

Figura 1: Exemplo da estrutura de dados de um estado.

Onde:

- `PROBLEM` contém os detalhes das tarefas. É um *array* de modo a facilitar o seu acesso (pois é imutável). Além disso, está também ordenado pelo tempo de início (para facilitar a penalização de sobreposições na `fast-heuristic`);
- `ALLOCATED` contém as tarefas já alocadas em turnos. Os turnos são representados como listas de tarefas de modo a optimizar a sua composição ao se poder aproveitar a lista anterior (estão representados do fim para o início). Por sua vez os turnos também são guardados numa lista, pois esta demonstrou ser a estrutura de dados primitiva (do Common Lisp) mais adequada para as mutações constantes resultantes da alocação de tarefas a turnos;
- `NON-ALLOCATED` contém as tarefas por alocar em turnos, sendo uma lista de tarefas, pois tal como em `ALLOCATED`, foi a estrutura de dados mais prática para remoções constantes;
- `COST` é o custo deste estado em relação ao seu predecessor;
- `TOTAL-COST` é o custo cumulativo do estado;

- `HEURISTIC` é o valor da função heurística neste estado, calculada na geração, de modo a se poder obter facilmente a heurística múltiplas vezes sem perda de desempenho (por exemplo na ILDS);
- `HEURISTIC-FUNCTION` é a função heurística a ser usada na geração;
- `HASH` identifica o estado com base nos turnos em `ALLOCATED`, de forma a acelerar os algoritmos em `procura.lisp` ao ser usado para identificar estados iguais.

Nós das procuras implementadas

Para modelar os nós das procuras implementadas usou-se a estrutura representada pelo exemplo da Figura 2.

```
#S(SEARCH-NODE
    :STATE #S(AFECTACAO-STATE
                ...)
    :PREDECESSOR #S(SEARCH-NODE
                    ...)
    :F 1080)
```

Figura 2: Exemplo da estrutura de dados de um nó de procura, para as procuras implementadas.

Onde:

- `STATE` contém o estado correspondente ao nó;
- `PREDECESSOR` contém o nó antecedente, para se obter o caminho para a solução (embora não seja relevante para o projeto foi usado para depuração);
- `F` contém o valor de um nó (para ser usado com a RBFS, a nossa abordagem alternativa).

Heurísticas

Não se descobriu nenhuma heurística que se conseguisse aproximar significativamente da solução ótima (nenhuma das heurísticas é admissível), nem nenhuma heurística com desempenho médio mas cálculo rápido. No entanto, desenvolveram-se as seguintes duas heurísticas:

greedy-allocation-heuristic

A `greedy-allocation-heuristic` aloca gananciosamente as tarefas que faltam ao turno que minimizar a sua duração (com a nova tarefa adicionada e ignorando a restrição 3). No fim calcula a duração dos turnos que alocou (agora com a restrição 3), e como o custo total já está incluído nos turnos originais do estado, subtrai o custo total do estado. A subtração do custo total faz com que as procuras que a usam se tornem procuras gananciosas. Apesar disso, tem resultados relativamente bons, sendo a melhor heurística descoberta e robusta em diferentes problemas. No entanto, o seu cálculo é relativamente lento, não conseguindo terminar o problema 4 (dos problemas de exemplo fornecidos) nas condições requeridas (menos de 5 minutos num computador comum).

fast-heuristic

A `fast-heuristic` é uma heurística simples que consiste em somar as durações das tarefas não alocadas. Além disso, de modo a incentivar a alocação de tarefas em conflito primeiro (pois correspondem a diferentes turnos, o que faz o custo aumentar mais cedo), soma adicionalmente os períodos de sobreposição de duas tarefas adjacentes (que têm de estar ordenadas por tempo de início). Este resultado é posteriormente multiplicado por um número arbitrário, de modo a convergir rapidamente para uma solução. Esta foi a primeira heurística desenvolvida no projeto, e embora se tenham desenvolvido heurísticas adicionais, as restantes apresentavam problemas de robustez em vários problemas, tendo por vezes um bom desempenho e por outras desempenho pior do que esta heurística simples. Devido à sua rapidez, é usada para problemas maiores (entre 257 a 768 tarefas).

Estratégias de corte

Foi implementada uma estratégia de corte simples que consiste em apenas guardar os melhores sucessores de cada nó, de modo a evitar uma ordenação de todos os sucessores em memória antes de os descartar, que em muitas situações era o suficiente para ficar sem memória disponível. Um sucessor gerado é mantido se o seu valor (custo + heurística) for menor do que a soma do valor do melhor sucessor (até essa altura) mais uma fração da diferença entre o pior e o melhor. Essa fração foi determinada como 1/16 para problemas até 512 tarefas e como 1/32 para problemas maiores. Esta estratégia de corte não é usada para a sondagem iterativa (e outras procuras não-informadas que foram testadas). A qualidade das soluções não pareceu ser afetada nos testes desempenhados.

Opções tomadas

Para a verificação de algumas restrições usou-se uma abordagem implícita, onde não são adicionadas pseudo-tarefas correspondentes a viagens, almoços ou pausas (os seus tempos são contabilizados conforme a verificação de condições). Esta abordagem tem a vantagem de causar um menor número de sucessores.

Para a ILDS funcionar em casos não binários, decidiu-se tomar uma abordagem onde o número de discrepâncias é decrementado em uma unidade até ser 0, pois pareceu ser uma extensão intuitiva do algoritmo original. Além disso, adiciona-se o custo total à heurística, pois tem melhores resultados do que a heurística apenas – pode-se inclusivamente considerar a junção dos custos como variantes dessas heurísticas. Por fim, decidiu-se apenas parar a ILDS após um certo período de tempo e não imediatamente a seguir à primeira solução encontrada, de modo a tentar encontrar soluções potencialmente melhores.

Para a melhor abordagem combinaram-se várias estratégias, de forma a garantir melhor qualidade da solução dentro das limitações computacionais. Usou-se o critério de que a procura tinha de terminar em menos de 3 minutos nas máquinas do grupo (exceto procuras de tempo limitado) e que tinha de correr com menos de 192MiB de memória no SBCL (para dar uma margem de segurança em relação à implementação que será usada pelo professor). Criaram-se vários perfis com base nos problemas de teste fornecidos:

- Até 192 tarefas: A*, `greedy-allocation-heuristic`, fração no corte de 1/16;
- De 193 a 256 tarefas: ILDS, `greedy-allocation-heuristic`, fração no corte de 1/16;
- De 257 a 512 tarefas: A*, `fast-heuristic`, fração no corte de 1/16;
- De 513 a 768 tarefas: ILDS, `fast-heuristic`, fração no corte de 1/32;
- Mais de 768 tarefas: Sondagem Iterativa.

Para a abordagem alternativa implementou-se a RBFS – originalmente tencionava-se implementar o SMA*, o que não conseguimos realizar. A RBFS foi a melhor alternativa mais simples encontrada.

Ideias para obter melhores resultados

O principal fator para obter melhores resultados seria uma heurística com melhor qualidade e rapidez de cálculo, que infelizmente não se conseguiu desenvolver. No entanto acreditamos que existe uma heurística melhor com uma ideia semelhante à da `greedy-allocation-heuristic`, talvez até mesmo admissível, ordenando os turnos e as tarefas de modo a não ser necessária complexidade quadrática. No entanto todas as variações tentadas nesta heurística resultaram no seu não funcionamento.

Para obter melhores resultados tendo em conta a memória limitada seria interessante implementar o SMA*, que era intuito original para a abordagem alternativa. A sua implementação é complicada pelo que se implementou antes a RBFS, que sofre de usar pouca memória e por isso tem um desempenho pior que o SMA*.

Resultados obtidos

Para medir o desempenho das várias procuras testaram-se todas as procuras implementadas no ficheiro `procura.lisp` fornecido, mais as que se implementaram (Sondagem Iterativa, ILDS e RBFS para a abordagem alternativa). Os resultados obtidos podem ser observados nas tabelas 1 a 6 (células por preencher significam que não terminou em tempo útil ou que esgotou a memória disponível):

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	1800	5	0	61	170	2,79
profundidade	1800	5	0	5	15	3
profundidade-iterativa	1800	5	0	168	575	3,42
ida*	1080	3	0	5	11	2,2
a*.melhor.heuristica	1080	3	0	5	21	4,2
a*.melhor.heuristica.alternativa	1080	3	0	12	30	2,5
sondagem.iterativa	1080	3	4384	3010422	9062104	3,01
ILDS	1080	3	5000	92733126	216	0
abordagem.alternativa	1080	3	0	6	11	1,83

Tabela 1: Resultados do problema do enunciado (5 tarefas).

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	-	-	-	-	-	-
profundidade	29520	82	8	82	3403	41,5
profundidade-iterativa	-	-	-	-	-	-
ida*	9808	25	600	82	1381	16,84
a*.melhor.heuristica	9740	25	1328	82	3334	40,66
a*.melhor.heuristica.alternativa	12778	32	32	237	1309	5,52
sondagem.iterativa	15005	39	4032	35607	2902784	81,52
ILDS	9661	25	5228	2440	24370	9,99
abordagem.alternativa	9808	25	576	83	1381	16,64

Tabela 2: Resultados do problema de teste 1 (82 tarefas).

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	-	-	-	-	-	-
profundidade	58320	162	84	162	13203	81,5
profundidade-iterativa	-	-	-	-	-	-
ida*	22720	56	4968	162	4324	26,69
a*.melhor.heuristica	22478	55	32692	162	11559	71,35
a*.melhor.heuristica.alternativa	27987	69	200	536	3637	6,79
sondagem.iterativa	30853	77	4280	11410	1921445	168,4
ILDS	22573	55	5820	163	5273	32,35
abordagem.alternativa	22720	56	4956	163	4324	26,53

Tabela 3: Resultados do problema de teste 2 (162 tarefas).

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	-	-	-	-	-	-
profundidade	87840	244	340	244	29890	122,5
profundidade-iterativa	-	-	-	-	-	-
ida*	35742	86	21440	244	11941	48,94
a*.melhor.heuristica	35012	85	185028	244	35410	145,12
a*.melhor.heuristica.alternativa	40498	96	600	250	5329	21,32
sondagem.iterativa	49290	125	4856	6125	1498301	244,62
ILDS	35141	85	22932	245	11617	47,42
abordagem.alternativa	35742	86	21140	245	11941	48,74

Tabela 4: Resultados do problema de teste 3 (244 tarefas).

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	-	-	-	-	-	-
profundidade	-	-	-	-	-	-
profundidade-iterativa	-	-	-	-	-	-
ida*	-	-	-	-	-	-
a*.melhor.heuristica	-	-	-	-	-	-
a*.melhor.heuristica.alternativa	77607	188	4136	487	18941	38,89
sondagem.iterativa	95801	239	5336	1892	925326	489,07
ILDS	-	-	-	-	-	-
abordagem.alternativa	-	-	-	-	-	-

Tabela 5: Resultados do problema de teste 4 (472 tarefas).

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
largura	-	-	-	-	-	-
profundidade	-	-	-	-	-	-
profundidade-iterativa	-	-	-	-	-	-
ida*	-	-	-	-	-	-
a*.melhor.heuristica	-	-	-	-	-	-
a*.melhor.heuristica.alternativa	-	-	-	-	-	-
sondagem.iterativa	136033	334	4696	694	537033	773,82
ILDS	-	-	-	-	-	-
abordagem.alternativa	-	-	-	-	-	-

Tabela 6: Resultados do problema de teste 5 (693 tarefas).

Como se pode observar nas tabelas 1 e 2, a **procura em largura primeiro** e a **procura em profundidade iterativa** apenas conseguiram resolver o problema do enunciado, pelo que são desadequadas para o projeto.

A **procura em profundidade primeiro** conseguiu resolver até ao terceiro problema (tabela 4), limitando-se a retornar a solução trivial correspondente a uma tarefa por turno. Não conseguiu resolver o problema 4 (tabela 5) pois não foram usadas estratégias de corte para as procuras não informadas e a memória foi esgotada.

O **A*, IDA*, ILDS e RBFS** ("abordagem.alternativa"), usando a `greedy-allocation-heuristic`, conseguiram resolver até ao problema 3 com soluções semelhantes, mas não conseguiram encontrar uma solução em tempo útil no problema 4. O **A***, usando a `fast-heuristic`, resolveu até ao problema 4, ficando sem memória no problema 5 (tabela 6). Por fim a **Sondagem Iterativa** resolveu todos os problemas, produzindo, no entanto, apenas soluções com um número de turnos igual a aproximadamente metade do número de tarefas.

Pode-se observar que até ao problema 3 o **A*** obtém as melhores soluções, mas nos seguintes excede o limite de tempo. Em alternativa ao A*, a **ILDS** aparenta obter soluções ligeiramente melhores que o IDA* e RBFS (a utilização de heurísticas não admissíveis produz efeitos diferentes conforme a procura, mesmo que esta pudesse garantir a solução ótima – A*, IDA* e RBFS). Escolheu-se por isso usar o **A*** e a **ILDS** na **"melhor.abordagem"**.

Com base nos resultados obtidos nas tabelas 1 a 6 e conforme descrito na subsecção "Opções tomadas", implementou-se uma estratégia para "melhor.abordagem" **híbrida** que obteve os resultados da tabela 7:

	Custo da solução	Número de turnos da solução	Tempo de procura (ms)	Expansões	Gerações	Fator médio de ramificação
Enunciado	1080	3	0	5	21	4,2
Problema 1	9740	25	1300	82	3334	40,66
Problema 2	22478	55	31736	162	11559	71,35
Problema 3	35141	85	22716	245	11617	47,42
Problema 4	77607	188	4156	487	18941	38.89
Problema 5	113086	269	12792	694	16254	23,42

Tabela 7: Resultados da estratégia "melhor.abordagem".

Conclusões

Dos vários tópicos necessários para a completude do projeto, um aspeto não foi cumprido: não se conseguiram implementar heurísticas de grande qualidade. As que se conseguiram implementar não produzem soluções ótimas nem aproximadamente ótimas. Esse facto em conjunção com parecerem muito simples não representa o tempo que foi despendido a pensar, implementar e testar múltiplas heurísticas.

Outro aspeto que não era especificamente requerido para o projeto mas que era do nosso interesse seria a implementação do algoritmo SMA* para a "abordagem.alternativa", o que não conseguimos fazer.