City Plan

Temática 3 - Problema de otimização

Ana Filipa Campos Senra – 201704077 | Andreia Gouveia Barreto – 201706430 | Cláudia Inês da Costa Martins – 201704136

MIEIC | IART | Ano Letivo 2020/21 | 31 Março de 2020

Definição do problema

Objetivo

Dado um conjunto de projetos de construção para uma cidade, decidir quais dos projetos disponíveis se devem construir e onde, de forma a maximizar a capacidade residencial e a disponibilidade de serviços públicos para os residentes.

Representação

A cidade é representada como uma grelha retangular.
As células dentro da grelha são referenciadas usando um par de coordenadas [linha, coluna]

Tipos de Projetos

Residenciais – plano de construção e capacidade

Utilitários – plano de construção e o seu tipo de serviço

```
.#
##
.#
```

Figura 1 – Exemplo de Plano de Construção

Distância Mínima de *Manhattan*

Utilidades estão disponíveis aos residentes de um projeto se a distância mínima de Manhattan entre o projeto residencial e ao projeto utilitário não for superior à distância máxima confortável para os residentes percorrerem a pé.

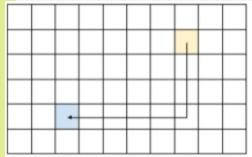


Figura 2 – Distância de Manhattan entre duas células.

Trabalho relacionado com referências a trabalhos encontrados na pesquisa

Foi realizado um trabalho de pesquisa extenso sobre o problema a solucionar. Este problema foi lançado na edição de 2018 do *Hash Code* da *Google*. A abordagem mais frequente foi a de **Divisão e Conquista** aliada a um **algoritmo ganancioso**:

1. https://github.com/kostyaHrytsyuk/CityPlan
2. https://news.itmo.ru/en/university_live/achievements/news/7499/
3. https://github.com/HofmannZ/hashcode-2018

Formalização do Problema

Representação da solução

O mapa da cidade é representado por uma *HashTable com as células ocupadas*. A *HashTable* terá como chaves pares (*Pair*) que representam a localização no Plano da Cidade (coluna, linha) e como valor o número do projeto presente na localização.

Função de Cruzamento

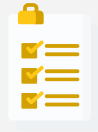
O cruzamento entre dois planos urbanísticos será feito trocando dois projetos compatíveis[1] entre os dois planos.

Função de vizinhança/ mutação

A mutação/ vizinhança consistirá em colocar um projeto aleatório no plano urbanístico numa posição aleatória.

Restrições rígidas

1. Todos os projetos têm estar na totalidade dentro da grelha da cidade (mesmo as células não ocupadas).
2. Células ocupadas dos projetos não se podem sobrepor (Uma célula ocupada e várias vazias ou várias vazias podem-se sobrepor).
3. Um projeto residencial tem de estar a uma distância não superior à de Manhattan de, pelo menos, um projeto utilitário.



Função de avaliação

Somatório do número total de serviços disponíveis a cada cidadão da cidade a uma distância mínima de Manhattan não superior à distância máxima confortável para os residentes percorrerem a pé.

[1]Compatíveis: Dois projetos que pelas suas características (tamanho e forma) sejam possíveis de ser trocados um pelo outro.

Implementação

Java

IntelliJ IDEA

Dados de Entrada

```
4 7 2 3
R 3 2 25
.#
##
.#
U 1 4 1
####
U 2 2 5
##
##
```

4 filas, 7 colunas, 2 é a máxima distância a pé, 3 Projetos
Projeto 0: tipo residencial, 3 filas, 2 colunas, capacidade de 25.
Células Ocupadas: [0, 1], [1, 0], [1, 1], [2, 1]

Projeto 1: tipo utilitário, 1 fila, 4 colunas, tipo de serviço 1.
Células Ocupadas: [0,0], [0,1], [0,2], [0,3].
Projeto 2: tipo utilitário, 2 filas, 2 colunas, tipo de serviço 5.
Células Ocupadas: [0,0], [0,1], [1,0], [1,1].

Figura 3 – Exemplo de Dados de Entrada

Dados de Saída

City Plan Optimization

Made by: Filipa Senra, Claudia Martins, Andreia Gouveia | IART | FEUP | 2020

InputFile: ./src/com/main/inputFiles/b_short_walk.in

Legend:

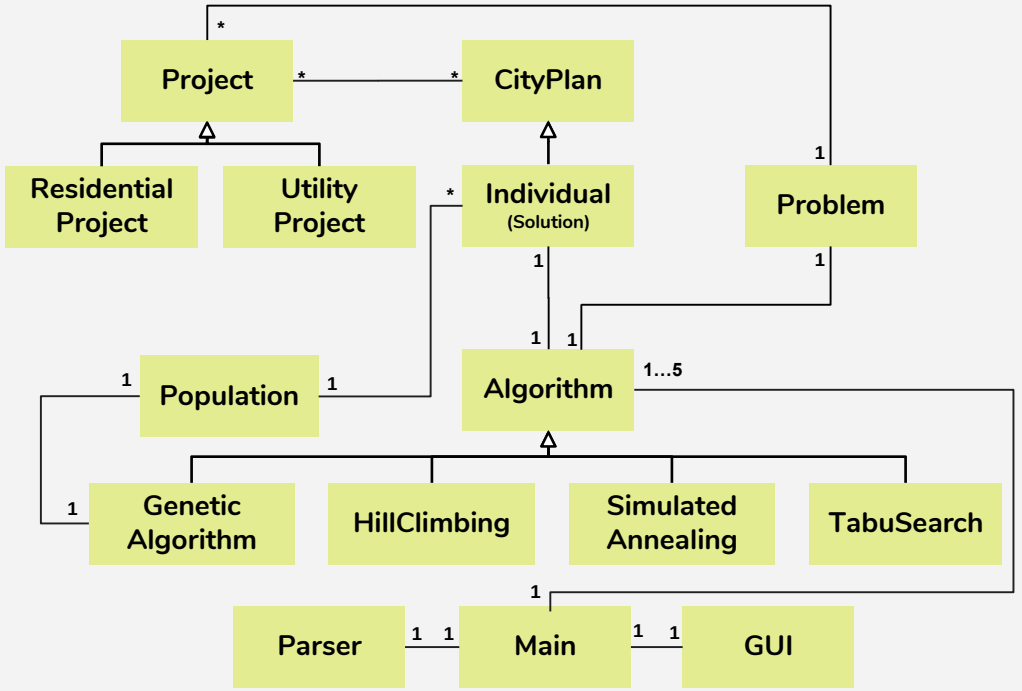
Cell without occupation	Cell with Residential Project	Cell with Utility Project

Genetic Algorithm

115	102	102	102	143	143	143	143	191	93
51	51	51	51	115	185	14	191	191	191
51	51	51	51	14	14	14	14	14	14

Figura 4 – Exemplo de Dados de Saída

UML de Classes

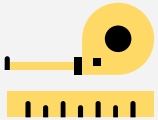


Abordagem

Função de Avaliação

Para cada projeto residencial com uma capacidade r colocada no mapa, o plano terá r pontos por todo tipo de serviço acessível aos moradores daquele prédio.

Um tipo de serviço é acessível aos residentes se existir um projeto utilitário que ofereça o serviço desse tipo a uma distância de Manhattan menor que a máxima distância a ser percorrida pelos moradores (indicada no ficheiro de input).



Operadores

Cruzamento (*crossover*) – É escolhido aleatoriamente quais os projetos (e de que progenitor) serão transmitidos ao descendente.

Mutação/Vizinhança (*mutate*) – Colocação de um projeto aleatório numa posição aleatória (retirando os projetos incompatíveis do projeto.

Elitismo – Os planos com melhor avaliação serão passados para a próxima geração (a percentagem é determinada pelo utilizador).

Seleção – Apenas os planos com melhor avaliação serão usados para a criação da nova geração.

Funções Auxiliares

Adicionar Projeto (*addProject*) – Adicionar Projeto no Local Indiciado (local definido pelas coordenados do canto superior esquerdo do Projeto).

Pré-Condição: O projeto tem de ser compatível com o local.

Retirar Projeto (*eraseProject*) – Retirar Projeto do Local Indicado (local definido pelas coordenados do canto superior esquerdo).

Pré-Condição: O projeto tem que existir no local indicado.

Algoritmos Implementados

Algoritmo Genético

O Algoritmo Genético utiliza uma População de Indivíduos. Os indivíduos são inicializados aleatoriamente.
Os melhores indivíduos passam para a próxima geração, e cruzam-se para a gerar próxima geração. Alguns indivíduos sofrem mutações, de modo a diversificar a população.

Hill Climbing

O Algoritmo *Hill Climbing* começa com um indivíduo aleatório. Este é tomado como a solução. Irá fazer diferentes mutações à solução. Quando encontrar um indivíduo com melhor avaliação, este passará a ser a solução. Este processo repete-se até chegarmos ao fim das iterações.

Simulated Annealing

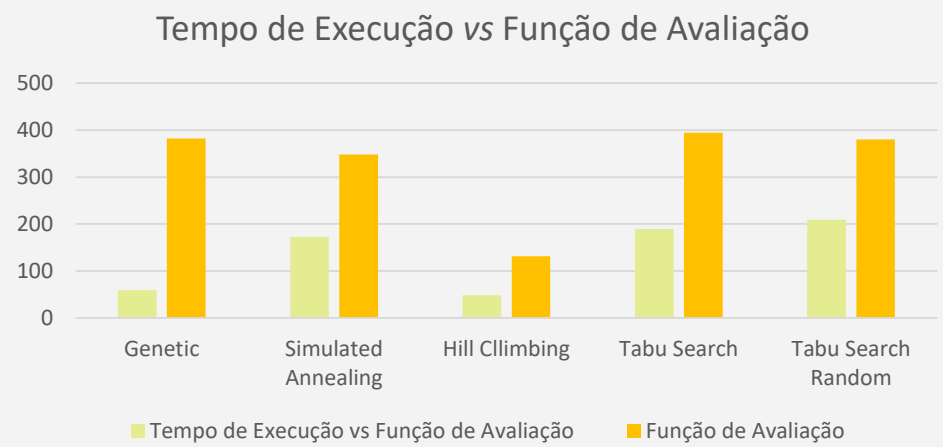
O Algoritmo *Simulated Annealing* é semelhante ao *Hill Climbing*. No entanto, aceita soluções que no momento parecem piores, com base na prioridade destas, mas que eventualmente poderão levar a uma melhor solução final.

Tabu Search

O Algoritmo *Tabu Search* é semelhante ao *Hill Climbing* ou ao *Simulated Annealing*, dependendo da versão. Difere-se destes, pois guarda as últimas N soluções, impedindo que soluções anteriores recentes sejam aceites (impede ciclos).

Resultados Experimentais

Todas as experiências podem ser consultados na Pasta “Experiências IART” em anexo



Gráfio 1 – Melhor resultados obtidos para o Tempo de Execução e Resultado da Função de Avaliação para o input “b_short_walk.in”.

Pudemos registar que o **Algoritmo Genético** permite a obtenção do **melhor resultado em menor tempo** em média. No entanto, o **melhor resultado** foi obtido pelo **Algoritmo *Tabu Search***.

É de notar que o tanto o Algoritmo ***Simulated Annealing*** como o ***Tabu Search Random*** obtém **bons resultados quando o número de iterações é baixo**. Isto deve-se, pois estes algoritmos permitem que sejam efetuados movimentos, que no momento parecem piores, mas que a longo prazo são melhores. Contudo, estas decisões são aceites no final das iterações, o que prejudica quando o número destas são elevadas.

Foi comprovado que o algoritmo ***Hill Climbing*** pode **ficar preso em máximos locais**, uma desvantagem deste método. Uma vantagem deste método é que é muito mais rápido que os restantes, podendo ser utilizado noutros problemas onde o problema de máximos locais não exista (ou seja pouco importante).

Função de Avaliação	Tempo de Execução (ms)	Niterações	Tamanho da População	Elementos que irão continuar para Próxima Geração (%)	Elementos que irão cruzar-se para gerar a próxima geração (%)	Elementos que sofram mutações em Cada Geração (%)	
120	835322040	400	100	0.1	0.5	0.5	1
110	335574760	200	100	0.1	0.5	0.5	2
120	1868994420	1000	100	0.1	0.5	0.5	3
115	225063820	200	50	0.5	0.5	0.5	
115	301343800	200	50	0.1	0.5	0.1	
110	503259200	200	50	0.5	0.5	0.5	4
100	278005320	200	50	0.1	0.5	0.5	5

Tabela 1 – Comparação do Desempenho do Algoritmo Genético para o input “a_exemple.in”.

1 3
A partir de um certo número de iterações, não são registadas melhorias na função de avaliação.

4 5
O Aumento do Elitismo não permite a diversificação necessária, refletindo-se no detrimento do resultado da função de avaliação.

1 2
O aumento da população permite p aumento da diversidade que se reflete no aumento do resultado da função de avaliação.

Conclusões

-01-

Objetivos

Todos os objetivos propostos foram alcançados.

Um ficheiro de output foi desenvolvido para melhor facilidade de leitura das soluções por parte do utilizador

-02-

Dificuldades...

... na adaptação do problema aos algoritmos de otimização.

... no tempo de execução longo devido à tipologia do problema.

Referências Consultadas e Materiais Utilizados

1. Kumar, Atul. *GeeksForGeeks*. s.d. https://www.geeksforgeeks.org/genetic-algorithms/ (acedido em 08 de 03 de 2020).

2. Rawat, Ujjwal. *GeeksforGeeks*. s.d. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/ (acedido em 13 de 03 de 2020).

3. Abraham, Joel. *GeeksForGeeks*. s.d. https://www.geeksforgeeks.org/simulated-annealing/ (acedido em 13 de 03 de 2020).

4. *CleverAlgorithms*. s.d. http://www.cleveralgorithms.com/nature-inspired/stochastic/tabu_search.html (acedido em 20 de 03 de 2020).

5. *Wikipedia*. s.d. https://pt.wikipedia.org/wiki/Simulated_annealing (acedido em 25 de 03 de 2020).