АНАЛИЗ НА РЕШЕНИЕТО НА **ПОБЕДИТЕЛЯ**

Няколко наблюдения по задачата:

1. Пространството на решения е много голямо. Трябва да фиксираме кои животни да използваме, в какъв ред и на каква позиция – това прави поне .
2. Ограничението по време не е много голямо. Т.е не може да се опитваме да оптимизираме по много сложен критерий.
3. В решението има значение редът на слагане на животните, т.е е по-трудно да правим промени по решението (например да разменим местата на две животни), без да го направим невалидно. Това е доста неприятно, защото влошава представянето на *hill climbing*[[1]](#one) и подобни.
4. Различни алчни алгоритми (от рода на: всеки път слагаме най-голямото по брой клетки животно на място, което ще ни донесе най-много точки) постигат доста добри резултати.

Това са общи наблюдения по задачата. Сега може да се съсредоточим върху различни начини да я направим по-лесна:

1. Ще оптимизираме само първите няколко хода. Нататък нещата стават прекалено трудни и едва ли може да очакваме да направим много в рамките на ограничението по време.
2. За оценка дали дадено начало е по-добро от друго може всяко едно от тях да довършим с алчен алгоритъм (например този, за който написах горе) и да сравним резултатите.

Тъй като искаме да оптимизираме само първите няколко хода, обхождане в широчина (*BFS*) по решенията е добър вариант (в широчина = по брой на животните в решението). Ще оставяме в опашката на *BFS* само *MAX* на брой най-добри решения, където *MAX* избираме според ограниченията (най-добри в смисъла на 2) ). Разбира се, това *BFS* ще работи докато не ни свърши времето, като отговорът ни ще е най-доброто нещо, до което сме стигнали (допълнено докъдето може с алчния алгоритъм).

Трябва да положим грижи да не попаднем на локален максимум, който да изхвърли от опашката много по-добро решение (по-добро в бъдеще; попадаме на този проблем, защото едно решение може да влезне като много копия в опашката с пренаредени ходове). Едно лесно решение, което се оказа по-добро от смятане на различни хеш-кодове или по-сложни оценъчни функции, е да добавяме с определена вероятност към опашката на *BFS* и по-лоши решения (реално не винаги да изхвърляме най-лошото, като опашката се препълни). Тази вероятност с времето трябва да намалява (ако е голяма получаваме нещо като *random search*; ако е малка – нещо като *hill climbing*). В моето решение имам температура на системата, която с времето намалява експоненциално, и вероятността по-лошо решение да влезе е функция на тази температура и доколко то е по-лошо (абсолютна разлика; тази функция трябва доста да се напасва, между впрочем). За повече информация, вижте [[2]](#two).

Това е всичко. Накратко решението може да се запише така:

*BFS + queue limit with greedy evaluation*.

*автор: Владислав Харалампиев*

**Препратки:**

[1] <http://en.wikipedia.org/wiki/Hill_climbing>

[2] <http://en.wikipedia.org/wiki/Simulated_annealing>