Solving The Words Search Problem

Ivan Kazmenko

St. Petersburg State University

Tuesday, July 5, 2011

Outline

Outline

Al Zimmermann's Programming Contests

- Held once or twice a year
- 17 contests so far since year 2001
- Typically lasts two or three months
- Each contest poses an optimization problem
- Participants run programs locally and submit answers
- Old Site: `http://recmath.org/contest`
- New Site: `http://azspcs.net`
- Our focus: contest #14, Words Search (Fall 2007)
  - 152 participants from 31 country
  - 26 596 total submissions

The Words Search Problem

- Fit as many words as possible into a 15×15 grid
- Words can go horizontally, vertically or diagonally in eight possible directions
- Word List:
  - ENABLE2K, a popular list for word games
  - 173 528 English words
- Subproblems:
  - There are 27 subproblems: `A`–`Z` and All letters
  - For the `A`–`Z` subproblems, only words containing the specific letter are counted
- Scoring System:
  - Each word is counted only once
  - For each word, the score is the length of the word
  - For each empty cell, the score is 1
  - You get *yours*/*record* points for each subproblem

- Subproblem: All letters
- Score: 4206
- Author: Vadim Trofimov

S	D	S	M	U	T	S	D	R	A	W	E	R	S	A
B	T	O	E	S	D	E	E	S	E	S	A	E	T	K
R	R	N	R	N	C	T	G	D	R	T	N	G	R	S
A	E	E	E	A	O	A	A	E	O	I	I	A	O	G
G	V	M	L	V	D	R	M	B	B	B	M	L	W	N
A	E	A	A	I	E	O	I	A	E	U	A	E	E	I
S	I	L	T	D	V	S	S	S	S	C	L	A	D	S
R	L	F	E	E	E	E	T	E	E	S	S	S	A	U
E	E	A	R	M	L	P	R	R	T	P	E	T	B	B
H	R	S	I	O	O	A	A	O	A	A	I	S	U	A
S	E	T	T	D	P	T	P	C	T	N	L	R	T	H
A	P	S	E	E	E	E	E	S	S	E	G	A	T	S
L	I	A	R	D	R	R	S	T	E	E	P	E	E	S
P	N	P	D	I	S	P	U	T	E	S	Y	A	R	D
S	S	O	F	T	E	N	S	C	R	A	M	P	S	S

- Subproblem: Letter ‘Q’
- Score: 1283
- Author: Ivan Kazmenko

N		G	N	I	Y	F	I	L	A	U	Q	E	R	P
O	O	P	A	Q	U	E	S	T	E	U	Q	O	R	C
N	D	E	R	I	U	Q	S	P	I	U	Q	E	E	R
U	E	S	D	E	T	A	U	Q	E	O	C	M	M	E
N	X	T	P	I	R	R	G	Y	U	O	U	A	A	A
I	C	A	I	I	S	E	S	S	N	S	C	S	R	C
Q	H	N	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
U	E	A	U	U	U	U	U	U	U	U	U	U	U	U
E	Q	Q	E	A	A	E	A	I	I	I	A	I	E	A
N	U	U	S	R	S	S	T	N	E	R	L	D	S	I
E	E	A	E	T	H	T	T	T	T	T	E	L	S	N
S	R	R	S	E	E	E	E	S	E	S	U	S	A	T
S	S	I	S	R	R	R	R	R	S	S		D	T	S
E		A	S	S	S	S	S	E	T	I	U	Q	E	R
S	I	N	C	O	N	S	E	Q	U	E	N	C	E	S

Outline

1. The Problem
   - Al Zimmermann's Programming Contests
   - The Words Search Problem
   - Example Grids
2. The Solution
   - Utilizing Classic Approaches
   - What We Can Change
   - Heuristics
   - Implementation Details
3. The Result
   - Benchmarks
   - Final Standings

A Few Classic Approaches To Combinatorial Optimization:

1. Full Search

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids

. . . way too many.

A Few Classic Approaches To Combinatorial Optimization:

1. Full Search

- Search Space $27^{15\times15} \approx 10^{322}$ possible grids

. . . way too many.

A Few Classic Approaches To Combinatorial Optimization:

2. Random Search

- Search Space — $27^{15 \times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Action — generate and score a random grid

Analysis:

- It takes much time to score a single grid
- We look at some random average grids

A Few Classic Approaches To Combinatorial Optimization:

2. Random Search

- Search Space — $27^{15 \times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Action — generate and score a random grid

Analysis:

- It takes much time to score a single grid
- We look at some random average grids

A Few Classic Approaches To Combinatorial Optimization:

3. Brownian Motion

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — always accept

Analysis:

- Recalculating the score is faster than scoring the whole grid
- We still look at some random average grids

A Few Classic Approaches To Combinatorial Optimization:

3. Brownian Motion

• Search Space	$27^{15\times15} \approx 10^{322}$ possible grids
• Objective Function	scoring function S
• Local Change	change a single cell
• Accepting Rule	always accept

Analysis:

- Recalculating the score is faster than scoring the whole grid
- We still look at some random average grids

A Few Classic Approaches To Combinatorial Optimization:

4. Hill Climbing

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $S_{new} \geq S_{old}$

Analysis:

- Recalculating the score is faster than scoring the whole grid
- We now find some good grids
- No way to leave a local maximum

A Few Classic Approaches To Combinatorial Optimization:

4. Hill Climbing

Search Space	$27^{15 \times 15} \approx 10^{322}$ possible grids
Objective Function	scoring function S
Local Change	change a single cell
Accepting Rule	accept if $S_{new} \geq S_{old}$

Analysis:

- Recalculating the score is faster than scoring the whole grid
- We now find some good grids
- No way to leave a local maximum

A Few Classic Approaches To Combinatorial Optimization:

5. Simulated Annealing

- Search Space — $27^{15 \times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Here, $\xi \in \mathcal{U}(0,1)$ (uniform distribution).

- When $S_{new} \geq S_{old}$, $P = (S_{new} - S_{old})/\mathcal{T} \geq 0$, so we always accept the change
- When $S_{new} < S_{old}$, $P = (S_{new} - S_{old})/\mathcal{T} < 0$,
  - When $\mathcal{T}$ is large, $|P|$ is small, so $\exp(P) \approx 1$ ($\approx$ Brownian Motion)
  - When $\mathcal{T}$ is small, $|P|$ is large, so $\exp(P) \approx 0$ ($\approx$ Hill Climbing)
  - In between, we try to get into a “good subspace”

A Few Classic Approaches To Combinatorial Optimization:

5. Simulated Annealing

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Here, $\xi \in \mathcal{U}(0,1)$ (uniform distribution).

- When $S_{new} \geq S_{old}$, $P = (S_{new} - S_{old})/\mathcal{T} \geq 0$,
  so we always accept the change
- When $S_{new} < S_{old}$, $P = (S_{new} - S_{old})/\mathcal{T} < 0$,
  - When $\mathcal{T}$ is large, $|P|$ is small, so $\exp(P) \approx 1$ ($\approx$ Brownian Motion)
  - When $\mathcal{T}$ is small, $|P|$ is large, so $\exp(P) \approx 0$ ($\approx$ Hill Climbing)
  - In between, we try to get into a “good subspace”

A Few Classic Approaches To Combinatorial Optimization:

6. Threshold Accepting

- Search Space — $27^{15\times15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $S_{old} - S_{new} \leq T$
- Schedule — gradually lower threshold T: $+\infty$ to 0

Here, the analysis is simpler.

- When $S_{new} \geq S_{old}$,
  we always accept the change
- When $S_{new} < S_{old}$,
  - When T is large, we usually accept ($\approx$ Brownian Motion)
  - When T is small, we usually reject ($\approx$ Hill Climbing)
  - In between, we try to get into a "good subspace"

A Few Classic Approaches To Combinatorial Optimization:

6. Threshold Accepting

Search Space	$27^{15\times 15} \approx 10^{322}$ possible grids
Objective Function	scoring function S
Local Change	change a single cell
Accepting Rule	accept if $S_{old} - S_{new} \le T$
Schedule	gradually lower threshold T: $+\infty$ to 0

Here, the analysis is simpler.

- When $S_{new} \ge S_{old}$,
  we always accept the change
- When $S_{new} < S_{old}$,
  - When T is large, we usually accept ($\approx$ Brownian Motion)
  - When T is small, we usually reject ($\approx$ Hill Climbing)
  - In between, we try to get into a "good subspace"

For this problem, simulated annealing works best.

What next?

What Can We Change?

- Search Space — $27^{15\times15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

What Can We Change?

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Experiments:

- Different starting and ending temperatures
- Different temperature switching mechanisms:
  - `for (T = 10.0; T >= 0.1; T *= 0.99999)`
  - Lower the temperature after x steps
  - Lower the temperature after either x steps or y accepts
  - *Increase* the temperature when we have too little accepts

What Can We Change?

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

No change: we already chose to do Simulated Annealing.

What Can We Change?

- Search Space — $27^{15\times15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Different modes:

- Consider just one random local change at a time (for high $\mathcal{T}$)
- Consider every possible local change, assign probabilities and choose a random one (for low $\mathcal{T}$)

What Can We Change?

- Search Space — $27^{15\times15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Possible local changes:

- Select a random word and write it in a random place (good for "hard" letters `J`, `Q`, `X`, `Z`)
- Assign probabilities to letters (based on word list, adjacent cells, etc.)

What Can We Change?

- Search Space — $27^{15\times15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Experiments:

- Don't give points for very short words, hoping to get them anyway

What Can We Change?

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Tradeoff:

- Possibly exclude some very good solutions, but:
- Increase speed of finding good solutions in what's left, and
- Obtain a "good subspace" with better average

What Can We Change?

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Experiments:

- For the "easy" letters, exclude "hard" letters and words containing them from consideration
- Find many good solutions; then, for future searches, exclude words that are not present in any of them
- Find many good solutions; assign probabilities to the letters used in local changes

What Can We Change?

- Search Space — $27^{15 \times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

Patterns:

- Manually recognize patterns, e. g. several equal letters in a certain row
  - Hard fix: do not permit changing
  - Soft fix: penalize score for changing
- Obtain patterns by merging previous good solutions

- Subproblem: Letter 'Q'
- Score: 1283
- Author: Ivan Kazmenko

N		G	N	I	Y	F	I	L	A	U	Q	E	R	P
O	O	P	A	Q	U	E	S	T	E	U	Q	O	R	C
N	D	E	R	I	U	Q	S	P	I	U	Q	E	E	R
U	E	S	D	E	T	A	U	Q	E	O	C	M	M	E
N	X	T	P	I	R	R	G	Y	U	O	U	A	A	A
I	C	A	I	I	S	E	S	S	N	S	C	S	R	C
Q	H	N	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
U	E	A	U	U	U	U	U	U	U	U	U	U	U	U
E	Q	Q	E	A	A	E	A	I	I	I	A	I	E	A
N	U	U	S	R	S	S	T	N	E	R	L	D	S	I
E	E	A	E	T	H	T	T	T	T	T	E	L	S	N
S	R	R	S	E	E	E	E	S	E	S	U	S	A	T
S	S	I	S	R	R	R	R	R	S	S		D	T	S
E		A	S	S	S	S	S	E	T	I	U	Q	E	R
S	I	N	C	O	N	S	E	Q	U	E	N	C	E	S

What Can We Change?

- Search Space — $27^{15\times 15} \approx 10^{322}$ possible grids
- Objective Function — scoring function S
- Local Change — change a single cell
- Accepting Rule — accept if $\xi < \exp\left((S_{new} - S_{old})/\mathcal{T}\right)$
- Schedule — gradually lower temperature $\mathcal{T}$: $+\infty$ to 0

After all experiments, refine the result running simulated annealing again with basic parameters.

Other techniques used for "hard" letters `J`, `Q`, `X`, `Z`:

- Start with an empty grid
- Try to put each possible word in each possible position in random order using Hill Climbing
- Continue until no such change increases the score
- After that, run the usual simulated annealing

Other techniques used for “hard” letters `J`, `Q`, `X`, `Z`:

- Start with a good grid
- Erase a random 3×3 or 4×4 rectangle
- Try to put each possible word in each possible position in random order using Hill Climbing
- Continue until no such change increases the score
- After that, run the usual simulated annealing

When You Run Out Of Ideas...

It's Time To Optimize!

The Data Structure: Trie

- Rooted tree
- Each edge has a letter assigned
- Each node corresponds to a string obtained by traversing the path from the root
- Some nodes correspond to words

Basic Implementation:

```
typedef struct node {
    int child [26]; // array index, -1 if none
    int word; // array index, -1 if none
};

node trie [MAXNODES];

int next (int curnode, int letter)
{
    return trie[curnode].child[letter];
}
```

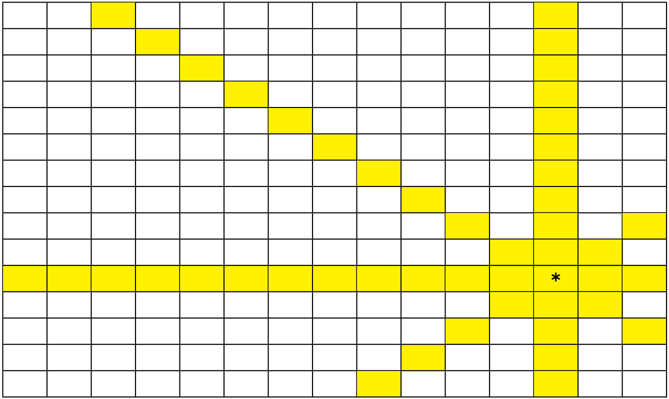
We need 108 bytes for each node.

Rescoring after changing a cell (x, y):

- Move in one of eight directions
- Starting from each visited cell, move in opposite direction and traverse a trie, looking for words containing cell (x, y)

The above procedure should be repeated twice:

- To decrease score for the old letter
- To increase score for the new letter

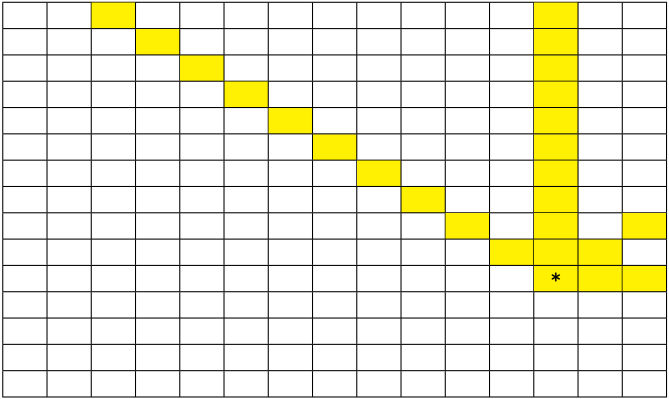


Optimization: For the subproblems, store only words containing specific letter

- The size of the trie is reduced

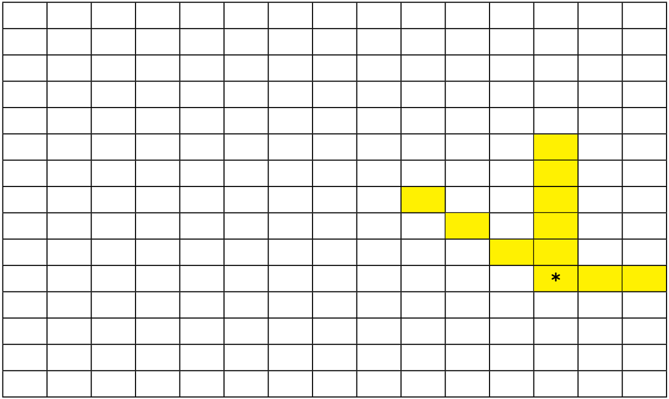
Optimization: Add reversed words to the trie

- Now we have to look only in four directions instead of eight
- Extra care should be taken for palindromes



Optimization: Add reversed prefixes to the trie

- Additionally, for each node we store the number of a “dual” node corresponding to the reversed string
- Now, all substrings and all reversed substrings of the given words are in the trie
- Now we can stop moving in a direction when the reversed prefix is not in the trie



Optimization: Build the trie using breadth-first search

- First goes the root, then all nodes corresponding to single-letter strings, etc.
- Siblings (children of a particular node in the trie) are adjacent and ordered lexicographically
- Now, instead of storing 26 indices, we can store one index pointing to the start of the children block and 26 bits indicating whether a particular child is present
- This greatly reduces the memory consumption and improves caching
- On the other hand, we now need $\approx \log 26$ operations to make a single transition instead of just one

An Example:

Suppose the trie stores just the strings “ac”, “aab”, “abc”, “abb” and “cba”. The table below demonstrates how it is stored.

Index	String	bits for c, b, a	start	+a	+b	+c
0	“”	101	1	1		2
1	“a”	111	3	3	4	5
2	“c”	010	6		6	
3	“aa”	010	7		7	
4	“ab”	110	8		8	9
5	“ac”	000				
6	“cb”	001	10	10		
7	“aab”	000				
8	“abb”	000				
9	“abc”	000				
10	“cba”	000				

Final Implementation:

```
typedef struct node {
    int bits, start, dual, word;
};

node trie [MAXNODES];
```

A node occupies only 16 bytes.

Final Implementation:

```
inline int next (int curnode, int letter)
{
    letter = 1 << letter;
    if (!(trie[curnode].bits & letter))
        return -1;
    int res;
    res = trie[curnode].bits & (letter - 1);
    if (!res)
        return trie[curnode].start;
    res = (res & 0x55555555) + ((res >> 1) & (0x55555555));
    res = (res & 0x33333333) + ((res >> 2) & (0x33333333));
    res = ((res + (res >> 4)) & 0x0F0F0F0F);
    res += (res >> 8) + (res >> 16) + (res >> 24);
    return trie[curnode].start + char (res);
}
```

Outline

Benchmarks:

On an average few minute Simulated Annealing run for All-letters:

- Trie nodes: 856 291
- Grids visited: 468 770 each second on an Athlon XP 3200+
- Average trie transitions for a single letter change: 101.5

Final Standings Top Ten:

Ivan Kazmenko	26.9069
Vadim Trofimov	26.8091
Fumitaka Yura	26.1996
Anton Maydell	25.8956
Mark Beyleveld	25.6342
Hanhong Xue	25.0067
Michael van Fondern	24.9398
Tudor-Mihail Pop	24.9023
Guido Schoepp & Klaus Müller	24.7280
Mikael Klasson	24.3040

The End