Chapter 5:

Results and discussion

5. Results and discussion

The implementations were evaluated on the set of road map images. Each file was compressed and the resultant size recorded. The total compressed size of all the files was computed and taken as the overall measure of compression effectiveness. Parameters of the compressors were changed, where applicable, to obtain best possible compression. Comparison between the methods were made, as well as with GIF as the benchmark. Section 5.1, 5.2 and 5.3 presents the results, for bitplane coding, PPM, and a combined method, respectively. Section 5.4 compares the methods in terms of compression effectiveness, memory requirement and execution time.

5.1 Bitplane coding

The images were compressed using the following neighbourhood templates:

1. Standard 1-norm – 10 to 16 bit templates
2. Standard 2-norm – 10 to 16 bit templates
3. JBIG default templates
    a. 10-bit template
    b. 13-bit template
    c. 16-bit template

Reflected Gray coding was also applied and the results compared with the case where it was not applied.

Table 5.1 shows the result when the images are compressed using bitplane coding for various template types, without reflected Gray coding. Table 5.2 shows the result with reflected Gray coding used.

Table 5.1: Results for bitplane coding (without reflected Gray coding)

file name**	template type *																
	10-1	10-2	10-J	11-1	11-2	12-1	12-2	13-1	13-2	13-J	14-1	14-2	15-1	15-2	16-1	16-2	16-J
a400600.raw	10519	10516	10619	10273	10365	10198	10286	**9892**	10145	10239	10018	10167	10051	10338	10156	10382	10132
b541541.raw	8928	8610	9550	8656	8481	8611	**8383**	8735	8492	8499	8690	8579	8792	8574	8804	8670	8733
c541541.raw	11237	10960	11906	10954	10823	10968	**10788**	11082	10857	10896	11141	10954	11221	11053	11255	11168	11214
d541541.raw	8879	8609	9336	8539	8474	8461	**8360**	8536	8388	8375	8506	8424	8586	8438	8636	8548	8615
e541541.raw	12871	12609	13428	12506	12459	12431	12372	12396	**12370**	12472	12539	12425	12543	12586	12611	12648	12592
f400400.raw	6011	5928	6237	**5918**	5934	5941	5931	6005	5969	6012	6103	6035	6200	6147	6290	6243	6234
g400400.raw	11185	11005	11793	10936	10923	10874	**10793**	10891	10819	10987	11039	10871	11177	11053	11336	11218	11203
h400400.raw	8383	8257	8702	8250	8275	8231	**8182**	8314	8220	8373	8448	8278	8545	8435	8693	8547	8575
i309356.raw	8820	8607	9603	8569	**8558**	8599	8578	8750	8644	8669	8872	8760	8885	8941	8992	8994	8956
j309356.raw	13418	13083	13741	13099	**13073**	13131	13076	13255	13195	13308	13486	13339	13684	13631	13905	13816	13816
k309356.raw	7476	7409	7593	7388	7314	7464	7377	7536	7423	**7257**	7500	7541	7574	7521	7622	7595	7573
l309356.raw	10979	10903	11331	10867	10831	10926	**10787**	11043	10838	10959	11189	10987	11296	11170	11411	11324	11316
m500500.raw	10485	9810	11474	9863	9711	9835	**9616**	9959	9740	9911	10093	9806	9819	9981	9898	9783	9820
n500500.raw	25378	24178	25956	24172	24224	24242	**23998**	24404	24125	24520	24767	24367	24834	24819	25195	24954	24912
o500500.raw	25226	24054	25745	24012	23961	23930	**23597**	24036	23701	24140	24318	23814	24388	24197	24704	24394	24464
total	179795	174538	187014	174002	173406	173842	**172124**	174834	172926	174617	176709	174347	177595	176884	179508	178284	178155

Note:

*Notation for template type: *order-type*

(e.g.: 10-1 means 10-bit 1-norm template, 10-2 means 10-bit 2-norm template, 10-J means 10-bit JBIG2 template)

** Size of compressed file in bytes

*** Best results for each file and overall are shown in **bold**.

Table 5.2: Results for bitplane coding (with reflected Gray coding)

file name **	template type*																
	10-1	10-2	10-J	11-1	11-2	12-1	12-2	13-1	13-2	13-J	14-1	14-2	15-1	15-2	16-1	16-2	16-J
a400600.raw	13578	13421	13894	13178	13183	13055	13058	**12740**	12968	13075	12881	12978	12840	13172	12977	13155	12945
b541541.raw	8350	8011	8994	8049	7868	7998	**7791**	8148	7891	7892	8136	7976	8235	7983	8226	8074	8152
c541541.raw	12879	12631	13573	12540	12478	12514	**12431**	12601	12449	12457	12662	12533	12767	12638	12832	12776	12824
d541541.raw	9656	9477	10146	9424	9374	9340	**9314**	9421	9356	9348	9449	9355	9523	9407	9576	9508	9606
e541541.raw	15374	14919	16229	14765	14694	14586	14502	14536	**14433**	14672	14714	14464	14679	14662	14741	14720	14660
f400300.raw	6203	**6095**	6510	6106	6096	6140	6116	6211	6188	6232	6357	6276	6471	6431	6586	6548	6515
g400400.raw	11477	11315	12194	11263	11266	11232	**11196**	11253	11246	11367	11449	11335	11641	11558	11843	11736	11675
h400400.raw	9739	9560	10363	9471	9557	**9383**	9472	9436	9443	9567	9602	9451	9720	9637	9895	9737	9776
i309356.raw	5645	5485	5936	**5417**	5445	5426	5446	5472	5424	5423	5494	5479	5583	5540	5549	5541	5528
j309356.raw	10072	9746	10588	9717	9721	9747	**9671**	9866	9733	9839	10015	9843	10106	10042	10278	10170	10199
k309356.raw	7182	7119	7414	7056	6974	7091	7000	7172	7025	**6784**	7017	7097	7035	6988	7077	7019	7025
l309356.raw	14944	14786	15451	14778	14679	14858	**14674**	15019	14776	14899	15234	14971	15400	15238	15559	15459	15464
m500500.raw	12761	12240	13775	12190	12150	12146	**12013**	12229	12052	12251	12388	12107	12141	12298	12242	12121	12143
n500500.raw	23194	21999	23529	21934	22064	22044	**21874**	22207	21926	22267	22515	22186	22546	22573	22878	22636	22577
o500500.raw	22220	21211	22918	21243	21117	21177	**20815**	21322	20963	21358	21586	21082	21691	21411	21943	21598	21687
total	183275	178015	191514	177134	176669	176737	**175373**	177633	175873	177431	179529	177133	180278	179578	182202	180798	180776

Note:

*Notation for template type: *order-type*

(e.g.: 10-1 means 10-bit 1-norm template, 10-2 means 10-bit 2-norm template, 10-J means 10-bit JBIG2 template)

** Size of compressed file in bytes

*** Best results for each file and overall are shown in **bold**.

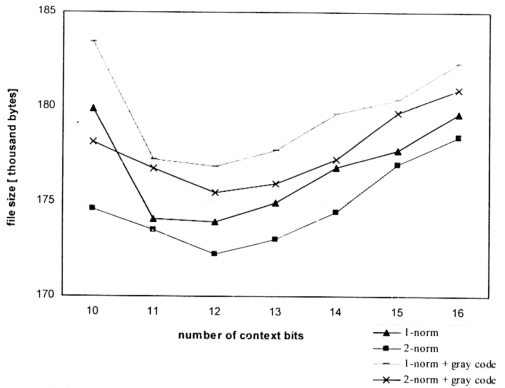

Figure 5.1: Plot of file size versus number of context bits used

From the results several observations can be made:

1. Figure 5.1 shows a plot of the file size versus the number of context bits used, for 1-norm and 2-norm templates. Starting from the order 10 context, it can be seen that the total size decreases until order 12. From then on, the total size increases again when the order increases. In all cases, the 12-bit template gives the best overall compression.
2. Figure 5.1 also shows that overall compression is better when Gray coding is not applied, for both 1-norm and 2-norm template. The 12-bit 2-norm template without Gray coding gives the best overall compression with a total size of 172,124 bytes.
3. The performance for the JBIG2 templates is summarized in Table 5.3. The best result is using the 13-bit template without Gray coding, giving the total size of 174,617 bytes. This is however not as good as the 172,124 bytes achieved by the 12-bit 2-norm template.

Table 5.3: Summary of results for JBIG2 templates

order	10	13	16
without gray coding	187014	174617	178155
with gray coding	191514	177431	180776

4. The original total size of the files is 3,080,740. The best result is 172,124 bytes, therefore

   percentage ratio = 100 x (172,124/3,080,740) = 5.6 %

   The files have been compressed to only 5.6% of the original total size, representing a reduction of 94.4%.

5. In GIF format, the total size is 305,052 bytes. Therefore,

   percentage ratio = 100 x (172,124/305,052) = 56.4 %

   The files have been compressed to 56.4% of the total size in GIF format, representing a reduction of 43.6%.

The 12-bit 2-norm template gives the best compression for these images. It is however difficult to explain theoretically why this is so, because the optimal template depends on the geometric structures in the image (Ageenko, 2001), which for the images are difficult to determine. We can only say that, based on the results this particular template gives the best estimation of the symbols' probability distribution of the test images compared with the other templates tried.

Gray coding does not improve the overall compression effectiveness. This is because the road map images do not contain regions with smoothly varying colours such as in a continuous-tone image. In a continuous-tone image, the regions of smoothly varying colours have neighbouring pixels which are close in value. By using Gray coding, two

different integers that are close together will differ by fewer bit positions compared to the normal binary representation (e.g. 127 and 128 differs by 1 bit position only in Gray code, but differs by 8 bit positions in normal binary). This reduces the complexity of each bitplane, resulting in improved compression (Rabbani & Melnychuck, 1992). For road map images, neighbouring pixels may differ significantly in value, so Gray coding is unable to guarantee the reduction of the complexity of the bitplanes, hence no improvement to the overall compression.

5.1.1 Conclusion

It can be concluded that, for bitplane coding:

1. The 2-norm template gives better compression, with the best result when using 12-bit context.
2. The use of Gray coding does not improve the overall compression
3. Compression using bitplane coding can achieve an overall reduction of 94.4% compared to the original files, and 43.6% compared to the GIF files.

5.2 Prediction by Partial Matching (PPM)

The following characteristics of PPM were investigated:

1. Update exclusion – comparing effectiveness of single counting and full counting
2. Neighbourhood template type – comparing the 1-norm and 2-norm templates
3. Escape probability – comparing the methods A, B, C and D
4. Maximum order – comparing values of 1 to 4
5. Frequency count scaling - various values when frequency count scaling is applied

The next sections present the results.

5.2.1 Update exclusion

Single and full counting was compared. The escape methods A, B, C and D were used. The other parameters were fixed to the following values:

1. 2-norm neighbourhood template
2. Maximum order = 3
3. Frequency count scaling value = 16384

Table 5.4 shows the results obtained. For each type of escape method, the best result for each file is marked in bold. As can be seen from the table, single counting gives better compression compared to full counting (except in a single case) for all the escape methods. This confirms the usefulness of applying update exclusion. For an explanation of why update exclusion can improve compression, see Moffat (1990).

Table 5.4: Results for PPM, to compare single and full counting

	method A		method B		method C		method D	
file name	single	full	single	full	single	full	single	full
a400600	**8584**	8628	**8734**	8788	**8720**	8727	**8654**	8674
b541541	**6525**	6579	**6653**	6719	**6646**	6667	**6560**	6599
c541541	**10826**	10890	**11043**	11115	**11031**	11039	**10928**	10960
d541541	**7651**	7718	**7809**	7912	**7794**	7822	**7717**	7761
e541541	**15392**	15437	**15614**	15657	15604	**15585**	**15497**	15504
f400400	**5382**	5468	**5488**	5612	**5467**	5522	**5412**	5481
g400400	**9700**	9736	**9823**	9873	**9794**	9808	**9708**	9728
h400400	**5647**	5749	**5783**	5925	**5756**	5819	**5683**	5767
i309356	**4767**	4806	**4849**	4898	**4866**	4871	**4805**	4828
j309356	**9186**	9249	**9298**	9374	**9292**	9311	**9226**	9265
k309356	**4552**	4617	**4645**	4727	**4631**	4656	**4549**	4591
l309356	**10500**	10620	**10699**	10839	**10658**	10709	**10566**	10649
m500500	**7117**	7310	**7339**	7634	**7235**	7369	**7157**	7316
n500500	**14303**	14716	**14893**	15407	**14625**	14811	**14387**	14671
o500500	**13916**	14243	**14424**	14809	**14215**	14350	**14033**	14251
total	**134048**	135766	**137094**	139289	**136334**	137066	**134882**	136045

**using 2-norm template, maximum order = 3, scaling = 16384*

5.2.2 Neighbourhood template

The standard 1-norm and 2-norm neighbourhood templates were compared. Escape methods A, B, C and D were used. The other parameters were fixed to the following:

1. Single counting
2. Maximum order = 3
3. Frequency count scaling value = 16384

Table 5.5 shows the results obtained. For each type of escape method, the best result for each file is marked in bold. As can be seen from the table, the 2-norm template gives better compression compared to the 1-norm template for all the escape methods. As explained in Section 5.1, we can only say that the 2-norm template is better at estimating the probability distribution, but it is difficult to theoretically explain why.

Table 5.5: Results for PPM, to compare 1-norm and 2-norm templates

	method A		method B		method C		method D	
file name	1-norm	2-norm	1-norm	2-norm	1-norm	2-norm	1-norm	2-norm
a400600	9456	**8584**	9600	**8734**	9595	**8720**	9526	**8654**
b541541	8363	**6525**	8482	**6653**	8489	**6646**	8386	**6560**
c541541	12361	**10826**	12570	**11043**	12569	**11031**	12456	**10928**
d541541	8888	**7651**	9046	**7809**	9029	**7794**	8947	**7717**
e541541	17061	**15392**	17272	**15614**	17273	**15604**	17147	**15497**
f400400	5853	**5382**	5960	**5488**	5948	**5467**	5889	**5412**
g400400	10395	**9700**	10527	**9823**	10509	**9794**	10421	**9708**
h400400	5996	**5647**	6135	**5783**	6109	**5756**	6040	**5683**
i309356	5870	**4767**	5949	**4849**	5972	**4866**	5912	**4805**
j309356	10852	**9186**	10957	**9298**	10974	**9292**	10900	**9226**
k309356	5148	**4552**	5239	**4645**	5224	**4631**	5153	**4549**
l309356	11973	**10500**	12165	**10699**	12146	**10658**	12051	**10566**
m500500	8893	**7117**	9106	**7339**	9013	**7235**	8917	**7157**
n500500	15288	**14303**	15853	**14893**	15608	**14625**	15352	**14387**
o500500	15470	**13916**	15956	**14424**	15776	**14215**	15576	**14033**
total	151867	**134048**	154817	**137094**	154234	**136334**	152673	**134882**

**single counting, maximum order = 3, scaling = 16384*

5.2.3 Escape method

The escape methods A, B, C and D were compared. The other parameters were fixed to the following values:

1. Single counting
2. Neighbourhood template type = 2-norm
3. Maximum order = 3
4. Frequency count scaling value = 16384

Table 5.6 shows the results obtained. The best result for each file is marked in bold. As can be seen from the table, method A gives the best compression in all cases except one. Overall, the best compression is given by method A, followed by D, C and B. As noted by Cleary & Witten (1984), it is difficult to justify the escape method theoretically, so we can only say that the method A performs best for the road map images.

Table 5.6: Results for PPM, to compare escape methods

file name	escape method			
	A	B	C	D
a400600	**8584**	8734	8720	8654
b541541	**6525**	6653	6646	6560
c541541	**10826**	11043	11031	10928
d541541	**7651**	7809	7794	7717
e541541	**15392**	15614	15604	15497
f400400	**5382**	5488	5467	5412
g400400	**9700**	9823	9794	9708
h400400	**5647**	5783	5756	5683
i309356	**4767**	4849	4866	4805
j309356	**9186**	9298	9292	9226
k309356	4552	4645	4631	**4549**
l309356	**10500**	10699	10658	10566
m500500	**7117**	7339	7235	7157
n500500	**14303**	14893	14625	14387
o500500	**13916**	14424	14215	14033
total	**134048**	137094	136334	134882

**single counting, 2-norm template, maximum order = 3, scaling = 16384*

5.2.4 Maximum order

Various values for the maximum order were investigated. The others parameters were fixed to the following values:

1. Escape method A
2. Single counting
3. Neighbourhood template type = 2-norm
4. Frequency count scaling value = 16384

Table 5.7 shows the results obtained. The test was done until order 4 only, because for orders larger than that, the increased memory requirements could not be met by the computer on which testing was done.

Table 5.7: Results for PPM, to compare various maximum order values

File name	maximum order			
	1	2	3	4
a400600	15959	10271	8584	**7797**
b541541	12630	8843	6525	**5482**
c541541	20650	13317	10826	**8648**
d541541	15950	9577	7651	**6056**
e541541	24946	17657	15392	**12534**
f400400	11267	6806	5382	**4300**
g400400	18185	11770	9700	**8237**
h400400	11783	6894	5647	**4623**
i309356	8553	6194	4767	**4395**
j309356	14119	11470	9186	**7853**
k309356	7208	5471	4552	**4106**
l309356	15863	12755	10500	**9214**
m500500	14997	9444	7117	**6013**
n500500	28690	16568	14303	**12292**
o500500	29819	16766	13916	**11973**
total	250619	163803	134048	**113523**

**single counting, 2-norm template, escape method A, scaling= 16384*

Figure 5.2 shows a plot of the total size against the maximum order. As can be seen, the higher the maximum order, the better the compression. This is because the larger context used enables better estimation of the symbol probability (Salomon, 2000). The maximum order of 4 gives a total size of 113,523 bytes.

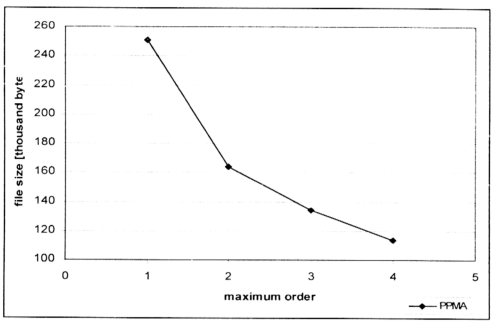

Figure 5.2: Plot of the total file size against the maximum order

5.2.5 Frequency count scaling

The effect of the value at which the frequency count scaling is done was investigated. The others parameters were fixed to the following values:

1. Escape method A
2. Single counting
3. Neighbourhood template type = 2-norm
4. Maximum order = 3
5. Frequency count scaling value = 16384

Table 5.8 shows the results obtained. The best result for each file is marked in bold.

Table 5.8: Results for PPM, to compare various frequency count scaling values

	frequency count scaling value											
file name	512	1024	2048	3072	4096	6144	8192	16384	32768	65536	262144	524288
a400600	8780	8631	8584	8577	**8573**	8575	8576	8584	8590	8596	8599	8599
b541541	7153	6765	6589	6539	6521	6508	**6507**	6525	6553	6591	6642	6642
c541541	11405	11026	10873	10841	10829	**10818**	**10818**	10826	10842	10853	10863	10863
d541541	8194	7856	7706	7672	7656	7646	**7645**	7651	7660	7671	7695	7695
e541541	15807	15479	15363	**15345**	**15345**	15353	15365	15392	15417	15431	15444	15444
f400400	5479	5389	5366	**5363**	5366	5368	5374	5382	5384	5385	5385	5385
g400400	9604	9585	9608	**9626**	9644	9658	9674	9700	9708	9709	9709	9709
h400400	5733	5654	5635	**5630**	5631	5636	5639	5647	5650	5655	5655	5655
i309356	5165	4927	4821	4789	4780	**4766**	4769	4767	4777	4783	4788	4788
j309356	9390	9262	9215	9209	9200	9192	9188	**9186**	9187	9188	9188	9188
k309356	4946	4693	4576	4546	4531	4531	**4526**	4552	4580	4593	4604	4604
l309356	10563	**10494**	10495	10496	10499	10500	10500	10500	10501	10501	10501	10501
m500500	7336	7167	7108	7100	7098	**7099**	7103	7117	7132	7139	7142	7142
n500500	14320	14213	**14202**	14218	14234	14255	14277	14303	14306	14306	14306	14306
o500500	14066	13908	**13875**	13878	13883	13894	13900	13916	13923	13925	13925	13925
total	137941	135049	134016	133829	**133790**	133799	133861	134048	134210	134326	134446	134446

Note:

* Best results for each file and overall are shown in **bold**.

** To test for scaling values of 65536 and above, the type for the arrays *freq [ ]* and *cum_freq[ ]* of the structure *ppm_prob_table* (Section 4.3.1.2.1) were changed to unsigned long

The total file size is plotted against the scaling value in Figure 5.3 for clarity. As can be seen, the lower the value that is used, the better the compression, until a certain point. For the values tested, 4096 gave the best compression overall. At less than 4096, the total file size increase again.

It is noted that, at the scaling value of 4096, the total file size is 133,790 bytes, while at the scaling value of 524,288 the total file size is 134,446 bytes. The reduction in size is only 134,446 – 133,790 = 656 bytes. This means that using a small scaling value improves the compression by only about 0.5% at the most. However, when the count scaling value is small, count scaling is performed more often, which may cause an increase in execution time. Since the improvement in compression is quite small, the tradeoff needs to be considered when deciding the value for count scaling.

Frequency count scaling helps the model to adapt more quickly to the changing symbol probability distribution (Moffat, 1990). Since count scaling has a small effect on the compression effectiveness here, it can be said that the symbol probability distribution is quite non-varying throughout the road map image.

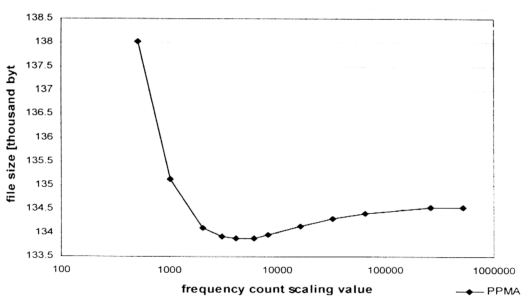

Figure 5.3: Plot of the total file size against the scaling value

5.2.6 Best settings

Based on the best settings above, the size of the compressed files is shown in Table 5.9 (PPM maximum order = 4). The total file size is 113,449 bytes, compared with the original total size of 3,080,740 bytes. Therefore,

$$\text{percentage ratio} = 100 \times (113{,}449/3{,}080{,}740) = 3.7\ \%$$

The files have been compressed to only 3.7 % of the original total size, representing a reduction of 96.3 %.

In GIF format, the total size is 305,052 bytes. Therefore,

$$\text{percentage ratio} = 100 \times (113{,}449/305{,}052) = 37.2\ \%$$

The files have been compressed to 37.2% of the total size in GIF format, which represents a reduction of 62.8 %.

Table 5.9: Results for PPM using best settings

file name	PPM maximum order =4	PPM variable max. order
a400600	7794	7794
b541541	5489	5489
c541541	8667	8667
d541541	6074	6074
e541541	12522	12522
f400400	4290	4290
g400400	8188	8188
h400400	4611	4611
i309356	4416	4780
j309356	7870	9200
k309356	4100	4531
l309356	9214	10499
m500500	6011	7098
n500500	12245	14234
o500500	11958	13883
total	113449	121860

*single counting, 2-norm template, escape method A, scaling= 4096

As shall be discussed in Section 5.4.2, the memory requirement for PPM becomes very large when the number of colours in the image increases. One way to constrain the memory requirement is to use a smaller maximum order value when the number of colours increases.

An example is shown in Table 5.9 (PPM variable maximum order), where the maximum order of 4 was used for files a400600 to h400400, while the value 3 was used for files i309356 to o500500.

In this case, the total file size is 121,860 bytes. Therefore

$$\text{percentage ratio} = 100 \times (121{,}860/3{,}080{,}740) = 4.0\ \%$$

The files have been compressed to only 4.0% of the original total size, representing a reduction of 96.0 %.

In GIF format, the total size is 305,052 bytes. Therefore,

$$\text{percentage ratio} = 100 \times (121{,}860/305{,}052) = 39.9\ \%$$

The files have been compressed to 39.9% of the total size in GIF format, which represents a reduction of 60.1 %.

5.2.7 Conclusion

From the results above, it can be concluded that:

1. The best overall compression was achieved using escape method A, with the 2-norm neighbourhood template.
2. Update exclusion was confirmed useful in improving the compression
3. The larger the value of the maximum order, the better the compression. However, for the experiments done, values up to 4 only were successfully tested, which gave the best compression overall compared to smaller values.
4. Reducing the value at which frequency count scaling is done improved compression performance (until a certain value), but the improvement was, at the most, only 0.5% (at the value of 4096 for PPM with escape method A).
5. Compression using PPM achieved an overall reduction of 96.3% compared to the original files, and 62.8% compared to GIF files, when the maximum order of 4 was used to compress the files.
6. The maximum order can be varied if the memory requirement has to be constrained. In this case, the compression effectiveness is slightly worse. For the example given, an overall reduction of 96.0% was achieved compared to the original files, and 60.1% compared to GIF files.

5.3 Combined method of bitplane coding and PPM

The combined implementation is another attempt to constrain the minimum memory requirement for the probability tables in PPM. In this method, PPM is used to compress the 4 lowest bitplanes, while bitplane coding is used for the rest of the bitplanes. The best settings from the results of Section 5.1 and 5.2 were used:

1. For PPM, the settings were: escape method A, 2-norm neighbourhood template type, maximum order of 4, single counting and frequency count scaling at 4096.
2. For bitplane coding: 12-bit 2-norm template, without applying Gray coding.

The results obtained are shown in the Table 5.10. The compressed files' total size for the combined approach is 123,474 bytes, compared with the original total size of 3,080,740 bytes. Therefore

$$\text{percentage ratio} = 100 \times (123{,}474/3{,}080{,}740) = 4.0\ \%$$

The files have been compressed to only 4.0% of the original total size, representing a reduction of 96.0 %.

In GIF format, the total size is 305,052 bytes. Therefore,

$$\text{percentage ratio} = 100 \times (123{,}474/305{,}052) = 40.5\ \%$$

The files have been compressed to 40.5% of the total size in GIF format, which represents a reduction of 59.5 %.

5.4 Overall comparison of the methods

In this section, the three methods implemented are compared with each other. Three important issues for a practical compression scheme are addressed: compression size, memory requirement and execution time.

Table 5.10: Best results for the various methods

	file name	bitplane coding	PPM max. order = 4	PPM variable max. order	combined
1	a400600	10286	7794	7794	7794
2	b541541	8383	5489	5489	5489
3	c541541	10788	8667	8667	8667
4	d541541	8360	6074	6074	6074
5	e541541	12372	12522	12522	12522
6	f400400	5931	4290	4290	4290
7	g400400	10793	8188	8188	8188
8	h400400	8182	4611	4611	4611
9	i309356	8578	4416	4780	5705
10	j309356	13076	7870	9200	8028
11	k309356	7377	4100	4531	4677
12	l309356	10787	9214	10499	10813
13	m500500	9616	6011	7098	7592
14	n500500	23998	12245	14234	15817
15	o500500	23597	11958	13883	13207
	total	172124	113449	121860	123474

5.4.1 Compression size

The results for each method from Sections 5.1, 5.2 and 5.3 are used to compare the compression effectiveness of the methods. The best results are summarized in Table 5.10. For PPM, two sets of results are given, one for maximum order of 4, another for variable maximum order (as in Section 5.2.6).

Based on Table 5.10, the compressed file size for each file is plotted in Figure 5.4 for the various methods. From the graph, it can be seen that:

1. PPM (with maximum order of 4) consistently achieved the best compression for all the files.

2. For files 1-8, the combined method achieved the same result as PPM (with maximum order of 4). This is expected since these files have the number of colours equal to or less than 16, so only PPM was actually applied. For files 9-15, the combined method is slightly worse than PPM.
3. The combined method was slightly worse overall compared to PPM variable maximum order. For files 1-8, the size is the same, for the same reason given in (2). For the other files, PPM variable maximum order gave better results, except for files 10 and 15.
4. Bitplane coding did not achieve as good compression as the other two methods for all the files, except files 5 and 12. For file 5, it is the best, while for file 12, it is better than the combined method.

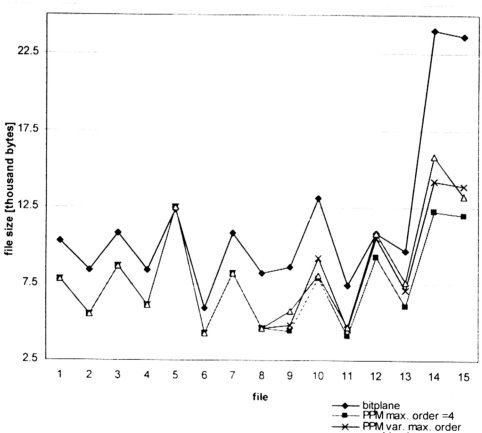

Figure 5.4: Plot of compressed file size of each file

With regards to the overall compression, the following summary can be made:

1. Bitplane coding - the total file size was reduced to 172,124 bytes. Compared to GIF, an overall reduction of 43.6% was achieved.
2. PPM with maximum order of 4 - the total file size was reduced to 113,449 bytes. Compared to GIF, an overall reduction of 62.8 % was achieved.
3. PPM with variable maximum order - the total file size was reduced to 121,860 bytes. Compared to GIF, an overall reduction of 60.1% was achieved.
4. Combined method – the total file size was reduced to 123,474 bytes. Compared to GIF, an overall reduction of 59.5% was achieved.

PPM achieved the best compression, followed closely by the combined method. Bitplane coding did not achieve as good compression as these two methods, but its size reduction was still better than GIF. Therefore, all three methods were able to compress the road map images, and compared to the GIF file format, does it more effectively.

5.4.2 Memory requirements

For the methods implemented, the demand for memory space is dominated by two items:

1. The buffer to store the data while performing compression and decompression
2. The probability tables used to store the models

The minimum memory requirement for each method (using the best results setting) when compressing or decompressing a file can be calculated as following:

1. Bitplane coding
    a. buffer size = image height x width x number of bitplanes

b. probability tables size = size of each probability table x number of contexts = $4x2^{12}$ = 16384 bytes

2. PPM

   a. buffer size = image height x width

   b. probability tables size = depends on the maximum order and number of colours used in the image (see Appendix D).

3. Combined method

   a. buffer size = image height x width x (1 + number of bitplanes for bitplane coding)

   b. probability table size

      - if the number of colours ≤ 16, see Appendix D
      - if the number of colours >16, the size of the probability tables is 7,060,405 bytes (see Appendix D, maximum order = 4, number of symbols = 16). This space can also be used by the probability tables of the bitplane coder, since they are not required at the same time.

Table 5.11 shows the minimum memory requirements for compressing each file. Several observations can be made:

1. For bitplane coding, the size of the probability tables is fixed to 16,384 bytes. The size of the buffer becomes the dominant factor. The size of the buffer depends on the image's dimensions and the number of bitplanes (which is determined by the number of colours in the image). However, for large images, this should not be a problem, because the implementation can be modified so that a fixed buffer size is used. Portions of the data can be read incrementally to the buffer for the compression process, instead of reading all the data at once (the sacrifice is, of course, some additional complexity).

Table 5.11: Calculated minimum memory requirements for the various methods

	bitplane coding			PPM (max. order = 4)			PPM (variable max. order)			combined method		
file name	buffer	prob. table	total	buffer	prob. table	total	buffer	prob. table	total	buffer	prob. table	
a400600	960000	16384	976384	240000	435479	675479	240000	435479	675479	240000	435479	675479
b541541	1170724	16384	1187108	292681	1143455	1436136	292681	1143455	1436136	292681	1143455	1436136
c541541	1170724	16384	1187108	292681	5152895	5445576	292681	5152895	5445576	292681	5152895	5445576
d541541	1170724	16384	1187108	292681	2568103	2860784	292681	2568103	2860784	292681	2568103	2860784
e541541	1170724	16384	1187108	292681	1741817	2034498	292681	1741817	2034498	292681	1741817	2034498
f400400	640000	16384	656384	160000	1143455	1303455	160000	1143455	1303455	160000	1143455	1303455
g400400	640000	16384	656384	160000	1741817	1901817	160000	1741817	1901817	160000	1741817	1901817
h400400	640000	16384	656384	160000	3682019	3842019	160000	3682019	3842019	160000	3682019	3842019
i309356	550020	16384	566404	110004	12560063	12670067	110004	697775	807779	220008	7060405	7280413
j309356	550020	16384	566404	110004	26750855	26860859	110004	1273844	1383848	220008	7060405	7280413
k309356	550020	16384	566404	110004	51583949	51693953	110004	2149325	2259329	220008	7060405	7280413
l309356	550020	16384	566404	110004	76516055	76626059	110004	2942919	3052923	220008	7060405	7280413
m500500	1250000	16384	1266384	250000	63069655	63319655	250000	2522780	2772780	500000	7060405	7560405
n500500	1250000	16384	1266384	250000	213232997	213482997	250000	6663525	6913525	500000	7060405	7560405
o500500	1250000	16384	1266384	250000	131124839	131374839	250000	4521540	4771540	500000	7060405	7560405

2. For PPM, the size of the probability tables is the dominant factor. With the maximum order fixed at 4, the memory size required becomes enormous when the number of colours in the image increases. At 32 colours (e.g. file n500500), about 213 Mbytes of memory space is required

3. For PPM, when the number of colours increases, the value for maximum order can be reduced to limit the memory requirement. For the example shown, the memory requirement is constrained to less than 7 Mbytes for an image containing 32 colours. However, the compression effectiveness is slightly reduced, as mentioned in Section 5.4.1.

4. For the combined method, the probability table is also the dominant factor. However, unlike PPM, the size of the memory required for the probability tables is bounded to about 7 Mbytes. This is because when the number of colours exceeds 16, the additional bitplanes are compressed using bitplane coding, whose probability tables do not require as much memory, and can share the same memory space since both coder's operate independently. The sacrifice is slightly poorer compression effectiveness, as mentioned in Section 5.4.1.

As a summary, bitplane coding has the smallest memory requirement. PPM has the most, especially when the number of colours increases and the maximum order is fixed. If the maximum order is reduced when the number of colours increases, the memory requirement can be constrained. The combined method also places a bound on the memory requirements even though the number of colour increases.

5.4.3 Execution time

Although the implementations were not optimized for speed, in terms of both the program coding and compilation, it would be useful to have a rough comparison of the different method's execution time. The compression and decompression time for each method was measured for all the files. A Pentium 4 1.8GHz computer with 192 MB RAM memory was used. The C language's *clock( )* function was used in the program to measure the time between the start and end of compression/decompression.

Three measurement samples were taken for each file and the average computed. The results are shown in Table 5.12 and also plotted in a graph, for both compression (Figure 5.5) and decompression (Figure 5.6).

Table 5.12: Compression and decompression time (in seconds)

		compression time [s]				decompression time [s]			
file name		bitplane coding	PPM max. order = 4	PPM var. max. order	combined method	bitplane coding	PPM max. order = 4	PPM var. max. order	combined method
1	a400600	0.240	0.579	0.579	0.604	0.161	0.579	0.579	0.578
2	b541541	0.281	0.729	0.729	0.755	0.193	0.729	0.729	0.750
3	c541541	0.281	0.890	0.890	0.912	0.193	0.890	0.890	0.932
4	d541541	0.281	0.776	0.776	0.823	0.188	0.776	0.776	0.807
5	e541541	0.276	0.786	0.786	0.807	0.198	0.786	0.786	0.802
6	f400400	0.156	0.427	0.427	0.422	0.109	0.427	0.427	0.453
7	g400400	0.161	0.453	0.453	0.474	0.115	0.453	0.453	0.443
8	h400400	0.161	0.510	0.510	0.516	0.115	0.510	0.510	0.521
9	i309356	0.141	0.641	0.266	0.578	0.094	0.625	0.266	0.615
10	j309356	0.146	1.171	0.276	0.547	0.110	1.318	0.287	0.542
11	k309356	0.141	1.495	0.297	0.558	0.094	1.515	0.323	0.537
12	l309356	0.135	1.985	0.323	0.609	0.104	4.078	0.328	0.589
13	m500500	0.281	2.016	0.646	0.901	0.219	3.073	0.615	0.901
14	n500500	0.292	34.370	0.672	0.938	0.219	47.302	0.740	0.922
15	o500500	0.297	12.620	0.636	0.932	0.218	18.714	0.688	0.922
	total	3.270	59.447	8.265	10.375	2.329	81.775	8.396	10.312

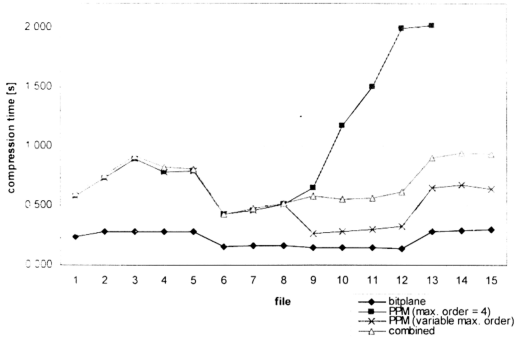

Figure 5.5: Compression time for each file

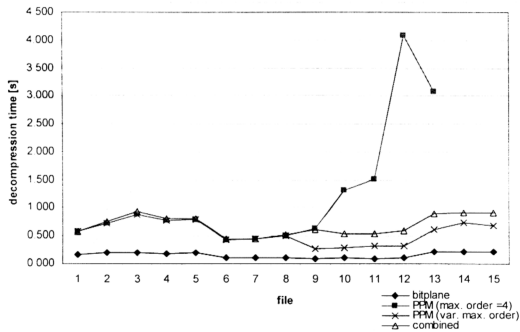

Figure 5.6: Decompression time for each file

For PPM with fixed maximum order of 4, the files 14 and 15 took a relatively long time to compress and decompress, and are not shown in the graph. This is because, for these files, the memory requirements are larger than the physical memory available in the

computer. Virtual memory had to be used, where the hard disk drive's storage space was used as random access memory. Reading from and writing to virtual memory on the hard disk (a condition called hard disk 'trashing') drastically slowed down the program's execution.

It can be seen from the results that:

1. Bitplane coding had the fastest execution time. Each file was compressed and decompressed in less than 0.3 seconds.
2. The execution time of PPM and the combined method were similar for files 1-8.
3. For files 9-15, PPM with a fixed maximum order of 4 became increasingly slower than the other methods. PPM with the variable maximum order was faster than the combined method.

As a summary, bitplane offers the fastest execution time, at least two times faster than PPM and the combined method.

5.5 Summary

All the three methods have been tested and their compression performance evaluated. Their compression parameters were adjusted to obtain the best possible compression. The results proved that the methods were able to compress the road map images. They were able to do so more effectively when compared to a commonly-used file format, GIF. Besides compression effectiveness, other practical issues such as memory requirement and execution time were investigated. As often is the case, there is a tradeoff between these factors.