Parallel Local Search for Permutations

Atle Riise

Collab workshop

12-13 April 2010

Motivaton

- Motivation
  - GPU (and similar) technologies are becoming increasingly accessible
  - How can it be used in local search?
- Case: Permutations, using the symmetric TSP as a test bench.

Local Search framework

IteratedLocalSearch

Input: initial solution s

1. *b = s*
2. *while (! stop)*
   a. *s = VND(s)*
   b. *Combine(s,b)*
   c. *s = Accept(s, b)*
   d. *s = Diversify(s)*
3. *Return b*

VND

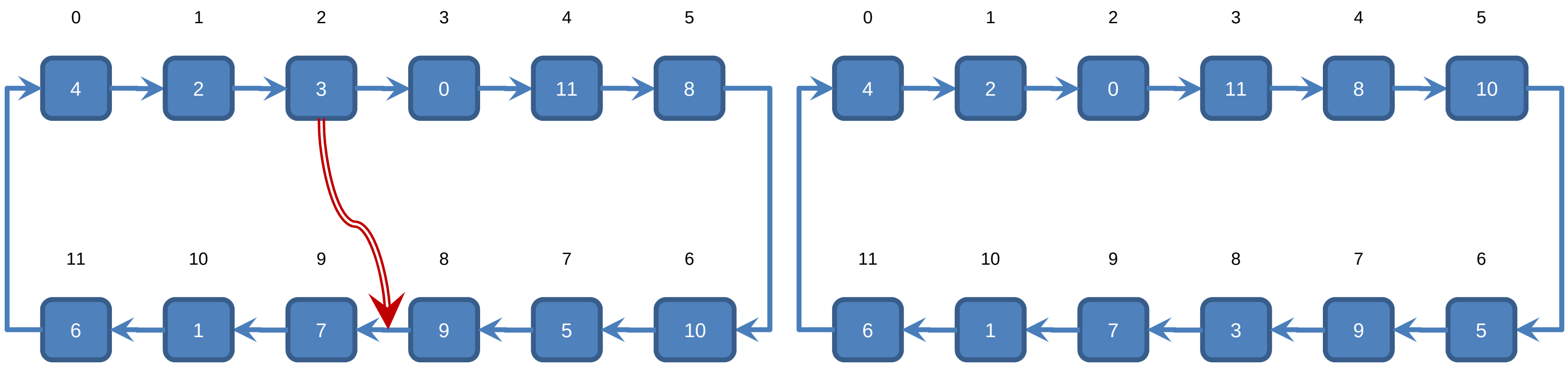
Input: initial solution s

1. *b = s*
2. *moveOp = twoOpt*
3. *while (moveOp != NULL)*
   a. *s' = Descent(s, moveOp)*
   b. *moveOp = SelectMO(s, s')*
   c. *s=s'*
4. *return s*

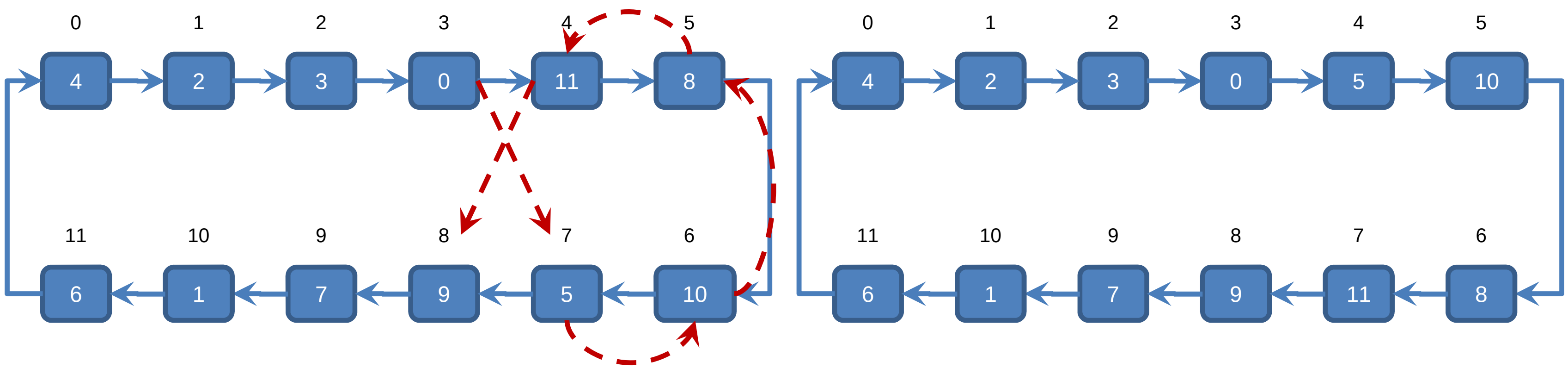
- Restart of ILS to avoid stagnation
  - Combination of solutions at each restart

Move operators

- Relocate, *O(n2)*



- Two-opt, *O(n2)*



Parallel Evaluation of neighbours

- In sequential LS, most of the computation time (>90%) is used in neighbourhood evaluation
- Obvious idea: Let each GPU kernel evaluate one neighbour (using a mapping from thread id to move id)
- Some authors have already done this
  - LS: Luong et al., Janiak et al.
  - GA/GP: Yu et al., Zhongwen and Hongzhi, Harding et al., Langdon and Banzhaf.

Parallel Evaluation speed-up

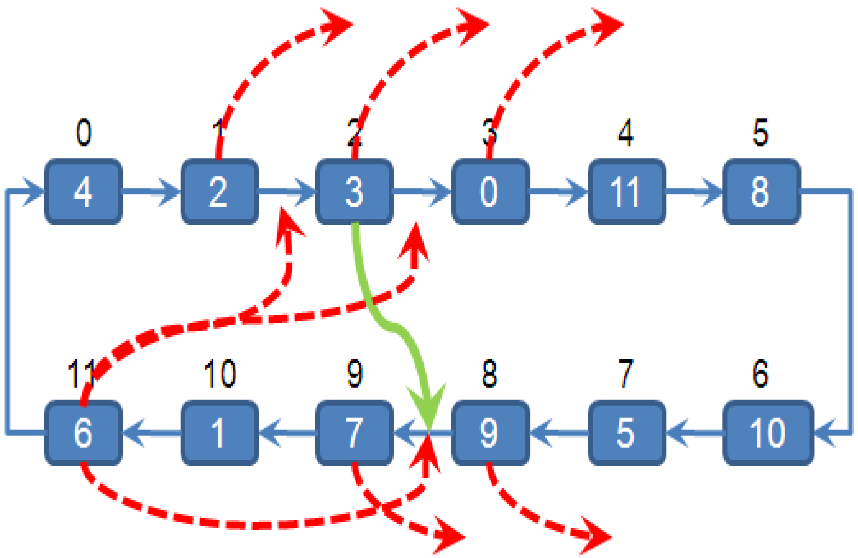
- GPU implementation
  - GeForce GTX 280
  - The best move is selected through reduction
  - Tested on a few cases. Speedup factor > 70 for all cases.
- Notes
  - Single precision on the GPU
  - Same search path, and both use delta evaluation
  - Parallel evaluation does not change complexity
  - So, it is still a point to reduce *large* neighbourhoods
  - Complex evaluations may not be implementable in parallel
    - Fast approximate parallel evaluation as neighbourhood reduction

So, what more can we do?

- In sequential search, one often use a limited neighbourhood exploration, e.g.:
  - ”First improvement”
  - NH-filtering (e.g. candidate lists). Typically using problem specific information.
- Our increased efficiency of NH evaluation reduces the need for such truncation
  - We can afford a more complete NH evaluation at each iteration

Combining independent moves

- We know **all** (or many of) the improving moves
  - Why only apply one (waste of computation effort)?
- However, cannot apply all based on the evaluation, since each evaluation assumes move independence.



Move independence and selection

- We have to select a set of moves whose evaluation is independent (objectives, constraints, ”modeling constraints”).
- Selecting a set of such independent moves from the set of all improving moves corresponds to the max. Weight stable set problem, which is NP-hard.
- Early in the search, we may have very many improving moves, and this complexity may be a problem.

Move independence and selection

- Two ways to go:
  - Congram et al. (”Dynasearch”), as well as Ergun et al:
    - Simplified dependency rules enables Dynamic Programming to select moves. Used in sequential search.
  - Our way
    - Exact dependency definition
    - Heuristic selection
- Note that the logic of selecting a set of basis moves applies equally well to best improvement sequential search.
    - However, in general, the GPU evaluation speedup enables best improvement search, and thus application of a maximum set of independent moves.

Move selection implementation

- Select independent improving moves using cudpp's compacting function
- Difficult to do selection on GPU due to "links". May be possible…?
  - Not a great case for parallelisation, unless the number of improving moves is large. However, would save copying memory to host.
- We ended up copying all improving moves to host, and using a sequential heuristic selection mechanism
- Then, since we are already on the host, we apply the few selected move sequentially. Then move on to evaluate the next iteration's neighbours on the GPU…

Similarities with VLNS

- Applying a set of independent (simple/basic) moves corresponds to applying a ”complex” move from a neighbourhood of ”all possible combinations of independent basic moves”.

- Such a neighbourhood is exponential in *n*, and a search with such neighbourhoods falls under the umbrella of Very Large Neighbourhood Search.

- However, we select our combined, complex, moves from a much smaller neighbourhood, based on only the *improving* basic moves.

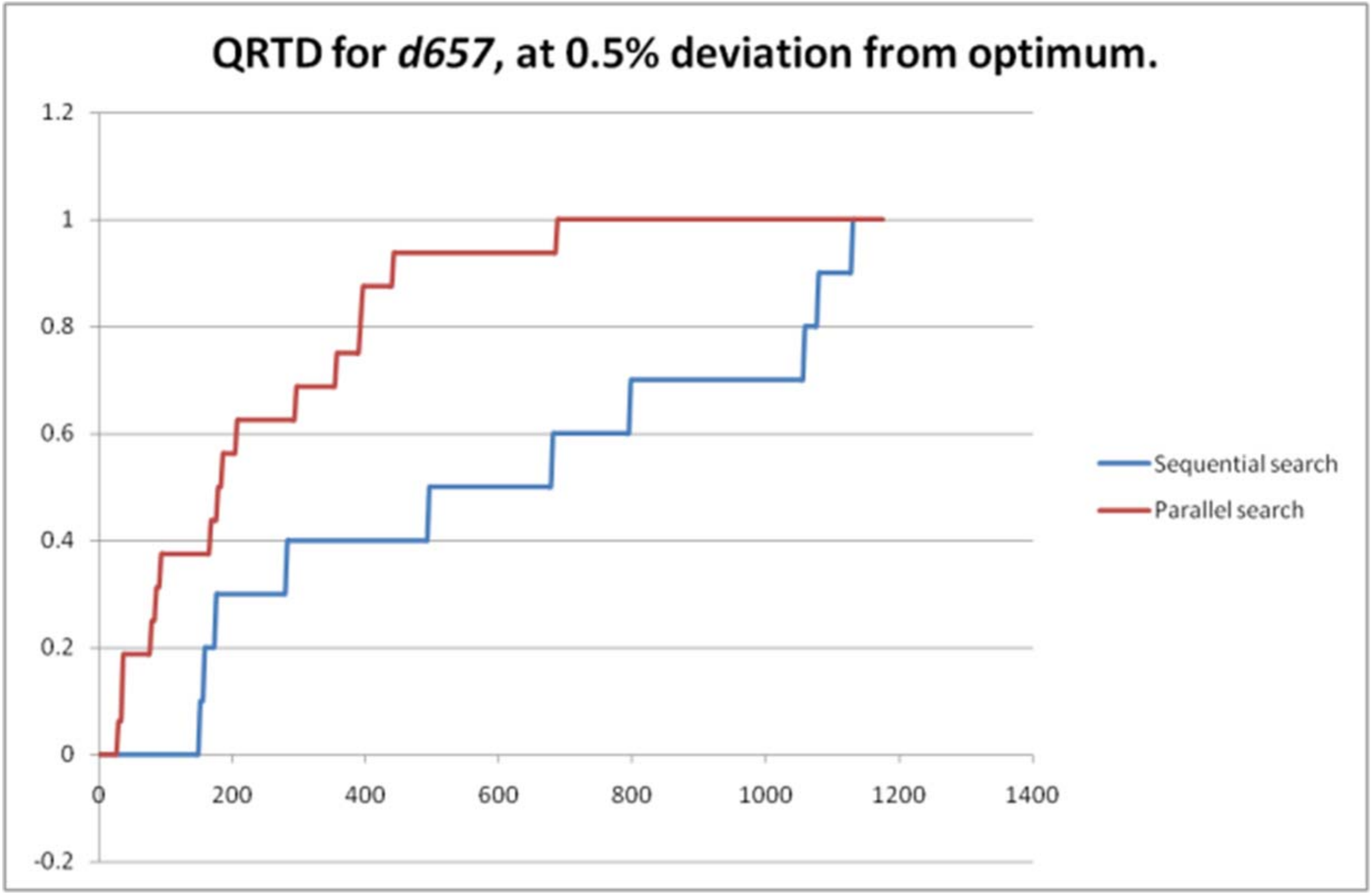
Parallel search speed-up

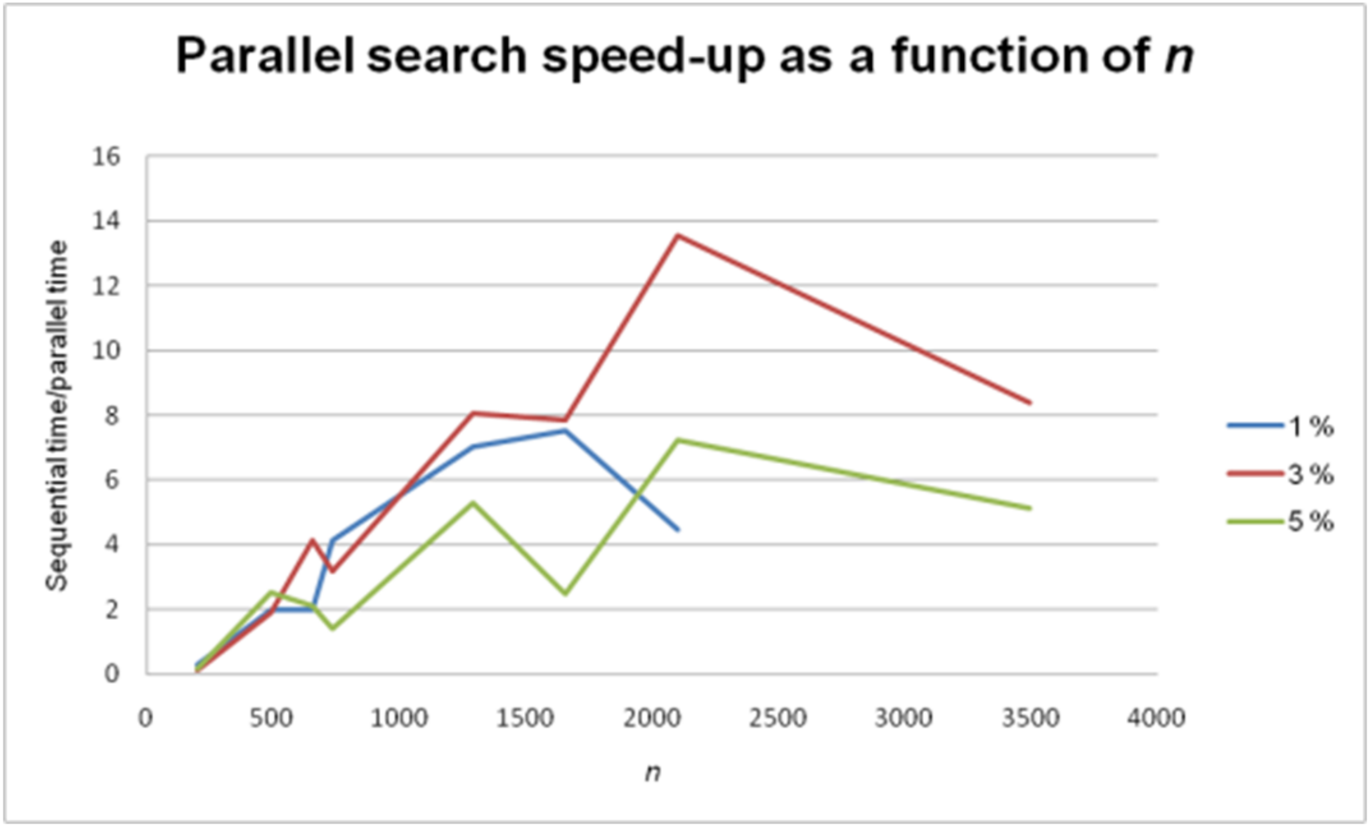
- Different paths through search space;
compare on cpu time

		Sequential Search				Parallel Search		
Case	**Run %**	**Mean**	**Min**	**Max**	**Run %**	**Mean**	**Min**	**Max**
d198	100 %	0.17316	0	0.546	100 %	0.558485	0.0468	1.263632
d493	100 %	34.48547	11.40367	71.88526	100 %	17.52043	5.850038	44.64749
d657	100 %	99.84438	45.6612	288.8808	100 %	50.46502	9.843663	150.0252
uy734	100 %	136.7368	72.4902	315.2604	100 %	33.11497	14.6016	68.99924
d1291	100 %	444.8726	119.4812	1052.788	100 %	63.43687	26.3266	118.3707
d1655	90 %	1083.724	363.1079	2208.149	100 %	144.1463	33.4776	478.764
d2103	100 %	724.8785	309.8024	1187.332	100 %	162.0577	8.6424	624.5869
nu3496	0 %	-	-	-	88 %	1586.162	608.0607	3537.784

Table 1: Mean time to reach a 1% deviation from the optimum value.

Example


QRTD for d657, at 0.5% deviation from optimum.
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
200
400
600
800
1000
1200
1400
Sequential search
Parallel search



The ratio between mean computation times used to reach different deviations from the optimal value, between sequential and parallel search.

Local Search framework

IteratedLocalSearch

Input: initial solution s

1. *b = s*
2. *while (! stop)*
   a. *s = VND(s)*
   b. *Combine(s,b)*
   c. *s = Accept(s, b)*
   d. *s = Diversify(s)*
3. *Return b*

VND

Input: initial solution s

1. *b = s*
2. *moveOp = twoOpt*
3. *while (moveOp != NULL)*
   a. *s' = Descent(s, moveOp)*
   b. *moveOp = SelectMO(s, s')*
   c. *s=s'*
4. *return s*

- Restart of ILS to avoid stagnation
  - Combination of solutions at each restart

Effect of combination

Early on in the search, the effect of re-combination of local optima does not seem to be important. As can be seen from Table 4, however, as the search approaches the optimum the combination has a positive effect on mean run times for all but one case.

Table 4: The effect of combination of local optima, at 1% deviation from optimal values.

		Without Combine				With Combine		
Case	Run %	Mean	Min	Max	Run %	Mean	Min	Max
d198	100%	0.941	0.031	2.075	100%	0.558	0.047	1.264
d493	100%	49.588	3.058	140.603	100%	17.520	5.850	44.647
d657	100%	75.073	24.274	160.274	100%	50.465	9.844	150.025
uy734	100%	69.932	18.205	155.111	100%	33.115	14.602	68.999
d1291	100%	69.371	21.512	195.562	100%	63.437	26.327	118.371
d1655	100%	366.812	61.604	828.656	100%	144.146	33.478	478.764
d2103	100%	100.414	9.376	467.782	100%	162.058	8.642	624.587
nu3496	100%	1 826.475	1 421.644	2 371.481	88%	1 586.162	608.061	3 537.784

References

- Luong, T.V., N. Melab, and E.-G. Talbi, *Parallel Local Search on GPU*, in 2009, DOLPHIN (INRIA Lille - Nord Europe).
- Janiak, A., W. Janiak, and M. Lichtenstein, *Tabu Search on GPU.* Journal of Universal Computer Science, 2008. **14**(14): p. 2416-2427.
- Yu, Q., C. Chen, and Z. Pan, *Parallel Genetic Algorithms on Programmable Graphics Hardware*, in *Advances in Natural Computation*. 2005. p. 1051-1059.
- Zhongwen, L. and L. Hongzhi. *Cellular Genetic Algorithms and Local Search for 3-SAT problem on Graphic Hardware*. in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. 2006.
- Harding, S. and W. Banzhaf, *Fast Genetic Programming on GPUs*, in *Genetic Programming*. 2007. p. 90-101.
- Langdon, W. and W. Banzhaf, *A SIMD Interpreter for Genetic Programming on GPU Graphics Cards*, in *Genetic Programming*. 2008. p. 73-85.
- Congram, R.K., C.N. Potts, and S.L.v.d. Velde, *An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem.* INFORMS J. on Computing, 2002. **14**(1): p. 52-67.
- Ergun, z., J.B. Orlin, and A. Steele-Feldman, *Creating very large scale neighborhoods out of smaller ones by compounding moves.* Journal of Heuristics, 2006. **12**(1-2): p. 115-140.