mad fast similarity search

Introducing the overlap engine

The overlap engine

Toolkit for similarity searches / similarity based overlap analysis of large sets: *doing a lot of similarity searches as fast as possible*

Speed: sustained 1.5 ns / comparison (~600M comp/sec) throughput of 1024 bit binary fingerprints on a single Amazon EC2 instance (c3.8xlarge).

Performance difference with an i7 CPU based laptop is about 4x.

What is it good for?

- Fast (multithreaded) descriptor generator (Chemical fingerprint, ECFP available with parametrization)
  0.5 min / million compounds on a c3.8xlarge instance
- Fast similarity search
  1.5 ns / comparison on a c3.8xlarge instance with multiple queries, various metrics
- Fast overlap analysis of large sets
  30 min / 1M x 1M comparison on a c3.8xlarge instance

Frontends

- Java SE API with usage examples
- Command line interfaces (sources available)
- Knime node
- Proof of concept web ui
  - For real time similarity search / drawing hints
  - Visualizing similarity based overlap of large (~1M) sets
  - Java SE standalone application with embedded Jetty server

Roadmap

- NOW: Use exported structures from DB (descriptor generation is fast: 0.5-2 min / M structures)
- Interest from users for out of the box distributed execution - should consider
- Speed up ordering / eliminating comparisons - "*never can be fast enough*"
- new JKlustor connection: use fast search/descriptor generation to back up similarity based clustering
- Higher level functional building blocks
  - Real time similarity search; backend for drawing hints
  - Overlap analysis in the range of M x M library sizes
  - Similarity based clustering speedup
  - Interactive clustering / structure explorer
  - Production-ready (G)UI

MFSS vs JChem/Screen

	MFSS	JChem/Screen
Goal	"*Doing a lot of similarity searches as fast as possible*"	"*Seamless integration of similarity into chemistry aware database deployments*"
Data source	Toolkit for standalone usage/custom server side integration; Working directly from structure files; Using local serialized data structures, no synchronization (*Regular update*)	Primarily DB backed storage; synchronization of changes without further effort; regeneration could be time consuming
Descriptors	CFP, ECFP, PFP, Screen3D, custom descriptors OOTB Can use multiple descriptors, multiple metrics	+BCUT; -Custom descriptors Such advanced use could be more intuitive
Functionality	Fast one/multiple query vs multiple targets Most similar, N of the most similars lookup Fast IO with compact serialized structures Fast concurrent descriptor generation	One query, only CFP cached, others very slow Only predefined cutoff Everything in DB; DB search/insert is on single thread on non-cached data, (CFP search executed in parallel) Integrated, synchronized solution
Architecture	Primarily API (toolkit) for various usages; simple frontends (CLI, KNIME, web ui prototypes) provided	java API/CLI/JChemBase/JChem Cartridge/KNIME/Instant JChem/JChem for Office/.NET as entry points
Collaboration	Backend API + components are licensed and closed source; frontend sources (CLI, KNIME) are disclosed; serving as usage examples	Backend API, closed source